Microsoft

GUIDA ALL'USO DI MS-DOS® QBasic™

Il QBasic è il potente e moderno linguaggio di programmazione che accompagna il sistema operativo MS-DOS. Imparare a programmare con QBasic è facile e piacevole grazie a un'interfaccia utente "amichevole", a efficaci strumenti di editing e debug e ottime funzioni di aiuto in linea.

Anche se non avete alcuna esperienza di programmazione, QBasic è la strada migliore per impararne i concetti fondamentali e iniziare a scrivere i vostri primi programmi in Basic. Il libro comprende:

Introduzione pratica e tutorial. Questa parte del volume fornisce una chiara spiegazione dei concetti che stanno alla base del progetto di un programma; istruzioni graduali per scrivere programmi in Basic, consigli sulle operazioni di debug e una guida di rapida consultazione per convertire i programmi da GW- Basic in QBasic di MS-DOS.

Programmi esempio. Moltissimi programmi di utilità pratica illustrano le corrette tecniche di programmazione.

Domande ed esercizi. Domande ed esercizi di ripasso sono stati appositamente pensati per migliorare e rendere più solido l'apprendimento di QBasic.

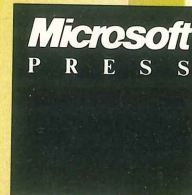
L. 50.000
IVA inclusa

ISBN 88-7131-039-X
9 788871 310398

GUIDA ALL'USO DI MS-DOS® QBasic™
MICHAEL HALVORSON
DAVID RYGMYR

GUIDA ALL'USO DI MS-DOS® QBasic™

Michael Halvorson
David Rygmyr

- Iniziare a programmare con il nuovo QBasic dell'MS-DOS
- Imparare i principi fondamentali e le tecniche avanzate di programmazione
- Scrivere programmi strutturati in Basic

L'edizione autorizzata

GUIDA ALL'USO DI

MS-DOS® QBasic™

Michael Halvorson
David Rygmyr

- Iniziare a programmare con il nuovo QBasic dell'MS-DOS
- Imparare i principi fondamentali e le tecniche avanzate di programmazione
- Scrivere programmi strutturati in Basic

Indice

Capitolo 7
Creare sottoprogrammi e funzioni proprie 147

Capitolo 8
Lavorare con grandi quantità di informazioni 175

Capitolo 9
Lavorare con le stringhe 219

CAPITOLO 1

Introduzione alla programmazione e a QBasic

PERCHÉ LA GENTE USA I COMPUTER?

Un computer non è altro che uno strumento al vostro servizio, in grado di aiutarvi a svolgere certi compiti in modo veloce ed efficiente. Il fatto stesso che abbiate comprato questo libro dimostra che pensate che un computer vi possa essere di aiuto, ma vi potreste chiedere a che cosa possa servirvi scrivere programmi.

Perché è utile imparare a programmare?

Dopo tutto oggi si possono trovare sul mercato migliaia di programmi, da quelli che vi aiutano a preparare la dichiarazione delle tasse a quelli che vi sfidano attraverso complessi giochi di avventura. E, proprio a causa della grande varietà di software in commercio, si potrebbe credere che il mercato sia già saturo: che esista, cioè, un programma per ogni esigenza.

Ma quest'idea è sbagliata.

La programmazione come strumento per risolvere problemi

Al di là delle generalizzazioni, i programmatori professionali hanno generalmente un occhio per il mercato. Per la maggior parte essi progettano programmi in grado di rispondere alle più ampie esigenze del mercato — infatti, da un punto di vista economico non avrebbe senso scrivere un programma per un ristretto gruppo di possibili acquirenti. Ciò significa che più specifiche sono le vostre esigenze, più improbabile è che troviate sul mercato un software già pronto in grado di soddisfarle.

Imparando a programmare sarete in grado di creare programmi personalizzati, in grado di soddisfare i vostri bisogni.

La programmazione come strumento di apprendimento

Un'altra ragione per imparare a programmare sta nel fatto che questo processo aiuta ad apprendere il funzionamento interno dei computer: si è in grado, così, di apprezzare la logica che sta alla base dei risultati che appaiono sullo schermo o in fase di stampa. Chi impara i fondamenti della programmazione è in grado di capire che cosa succede "dietro le scene", in modo molto simile a un architetto che osservando l'esterno di un palazzo può capire l'organizzazione delle travi di acciaio, delle tubature e delle altre strutture di supporto. Questa familiarità consente di capire le vere potenzialità di un computer; per questa ragione l'insegnamento della programmazione è alla base di qualsiasi corso di studi nel campo dell'informatica. Inoltre, acquisendo una conoscenza di fondo si è in grado di parlare il linguaggio dei professionisti del settore, attraverso l'uso di termini e concetti appropriati.

La programmazione come divertimento

L'ultima ragione per imparare a scrivere programmi per conto proprio è che programmare può essere molto divertente, sia per i principianti sia per i programmatori esperti.

Ci sono, dunque, molte buone ragioni per imparare a programmare. Per cominciare, cerchiamo di imparare un po' di terminologia; per esempio, che cos'è un programma?

Programmazione personalizzata

Le vostre esigenze potrebbero essere molto semplici; forse ciò di cui avete bisogno sono solamente dei programmi organizzativi in grado di semplificarvi l'esistenza. QBasic permette di creare programmi molto semplici o molto particolareggiati, a seconda dei bisogni dell'utente. Eccovi un elenco di utenti che potrebbero trarre vantaggio dal produrre i propri programmi:

- Un tifoso di calcio, che vuole raccogliere dati statistici su squadre e giocatori e poter essere in grado di ritrovare immediatamente informazioni di particolare interesse.
- Un appassionato di video giochi, che vuole crearsi un proprio gioco di avventura.
- Un manager o un piccolo imprenditore, che vuole registrare appuntamenti e date importanti per poi essere avvisato automaticamente quando questi si avvicinano.
- Uno studente o uno scienziato, che vuole eseguire calcoli matematici particolari.
- Il proprietario di una casa, che vuole mantenere un registro delle entrate e delle uscite.
- Un medico, che vuole raccogliere dati su pazienti, ricette e spese senza essere limitato dagli strumenti tradizionali della contabilità .

CHE COS'È UN PROGRAMMA

Per poter funzionare in modo utile un computer dev'essere completo di due componenti: hardware e software.

- Il concetto di hardware definisce le parti fisiche del computer, quelle che si possono vedere e toccare come la tastiera o lo schermo.
- Il concetto di software è un po' più elusivo: è l'intelligenza che controlla l'hardware, permettendo l'effettivo impiego del computer.

È impossibile usare l'uno senza l'altro; l'hardware senza il software è come un giradischi senza dischi.

Un programma per computer viene chiamato software: questo non è altro che un insieme di istruzioni che agiscono sul computer facendogli eseguire un certo compito, per esempio un calcolo matematico o un'elaborazione di testo. Un programma può essere composto da una sola istruzione, ma può contenerne anche centinaia o migliaia.

I computer: sono i buoni o i cattivi?

Il cinema e la televisione hanno spesso sottolineato come l'uso dei computer riduca l'occupazione, renda la vita più difficile e sia sul punto di dominare il mondo; insomma non sembra proprio l'immagine di uno strumento da acquistare per la casa o il lavoro, tanto meno da usare o programmare!

Nonostante la cattiva pubblicità i personal computer sono ormai entrati in molti luoghi di lavoro e in molte case, rendendo la vita più semplice e produttiva grazie alla capacità di automatizzare l'esecuzione di molte attività. L'opinione della gente su queste strane macchine elettroniche sta finalmente cambiando; quelle che una volta erano macchine in grado di produrre solamente una gran quantità di incomprensibili schede perforate e che occupavano un bel po' di spazio in un ufficio, sono oggi macchine che occupano poco spazio su una scrivania e mostrano immagini utili e divertenti. I computer sono ormai diventati parte integrante di ogni attività aziendale, di comunicazione, di intrattenimento e di ricerca scientifica; facili da usare e sempre più potenti, sono finalmente diventati quegli strumenti utili che fino a pochi anni fa si potevano soltanto sognare.

COME SI SCRIVE UN PROGRAMMA PER COMPUTER?

I primi importanti passi nel processo di programmazione sono la preparazione e l'organizzazione. Due domande possono aiutare a organizzarsi:

1. Che cosa si vuole che il programma faccia?
2. Quali passi deve compiere il programma per eseguire il suo compito?

L'insieme dei passi viene detto algoritmo. Un algoritmo non è un programma in se stesso, ma soltanto una raccolta di note ordinate che descrivono ciò che il programma farà in corrispondenza a ogni passo. Per fare un esempio, quando si usa una ricetta si usa in realtà un algoritmo: cucinare una torta implica infatti un processo da seguire passo passo se si vuole ottenere un risultato utile (e saporito!). Dal momento che un algoritmo è uno strumento per la risoluzione di problemi che viene progettato per aiutare in fase di scrittura del programma, non esiste un modo giusto o sbagliato da seguire nella sua creazione. Alcuni programmatori scrivono un algoritmo sotto forma di elenco ordinato di azioni, altri lo disegnano graficamente (grafico a flusso) e altri ancora organizzano semplicemente le diverse azioni a mente. Qualunque sia la vostra scelta, è importante che cominciate a pensare al vostro programma in termini generali molto prima di sedervi davanti al computer.

Il linguaggio di programmazione Basic

Una delle ragioni per progettare un programma e risolvere i problemi in anticipo sta nel fatto che i computer non possono riconoscere molte parole ed è necessario dar loro istruzioni in un linguaggio che siano in grado di capire.

Il Basic è uno dei molti linguaggi per personal computer disponibili. Come per tutte le lingue anche il linguaggio per computer ha un suo "vocabolario" e un suo insieme di regole. Inoltre, ogni linguaggio presenta dei punti forti e dei punti deboli: alcuni sono più efficienti nell'eseguire certi compiti, altri sono più semplici da usare. Il Basic è un eccellente linguaggio di programmazione perché è potente, flessibile e soprattutto facile da usare.

Imparare a "parlare" il Basic o qualsiasi altro linguaggio di programmazione è molto simile a imparare a parlare una lingua straniera; il Basic ha un suo vocabolario e un insieme di regole da seguire nell'uso del vocabolario stesso. Fortunatamente il vocabolario del Basic consiste di meno di 200 parole (dette *parole chiave*) e le regole (dette *regole di sintassi*) per l'uso delle parole chiave sono relativamente semplici.

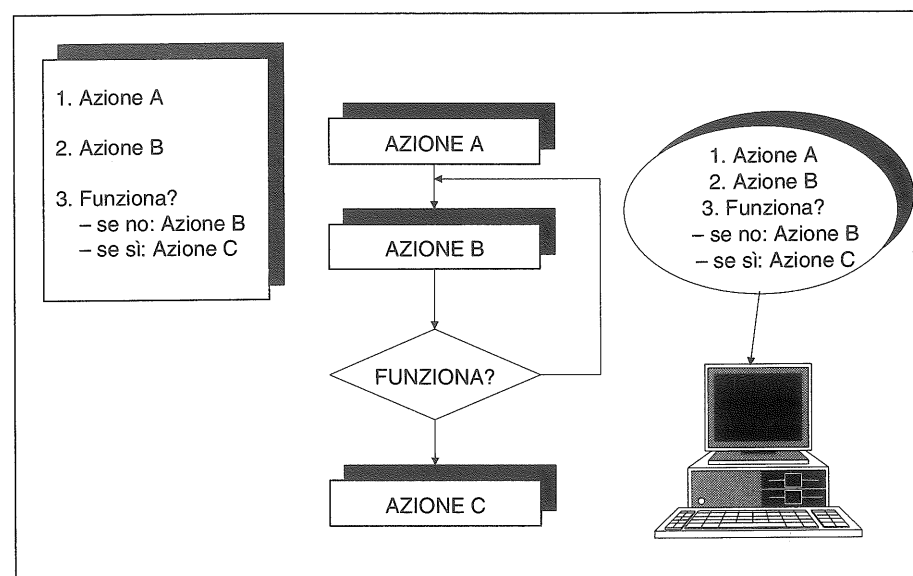



Figura 1-1. *Ci sono vari modi per creare un algoritmo.*

MS-DOS QBasic Interpreter

Prima di poter cominciare a digitare istruzioni in Basic il computer dev'essere dotato di un software in grado di capirle. Questo è il compito del programma MS-DOS QBasic Interpreter (QBasic) di cui è dotata la versione 5 di MS-DOS: esso legge le istruzioni QBasic e le traduce in una forma che il computer è in grado di usare. La parola "interpreter" è particolarmente appropriata in quanto MS-DOS QBasic Interpreter interpreta letteralmente ogni enunciato digitato, controllando che non ci siano errori nelle istruzioni inserite, che le parole chiave non siano state usate in modo sbagliato e che le regole di sintassi siano state rispettate.

QBasic fornisce anche un ambiente di programmazione nel quale lavorare. Esso è perciò un pacchetto completo che permette di scrivere programmi e di eseguirli immediatamente; con QBasic è possibile salvare i programmi creati su disco, stamparne le istruzioni e disporre di una guida on-line quando necessario. L'ambiente di programmazione QBasic dispone anche di molte funzionalità di elaborazione testo che semplificano la scrittura e la modifica dei programmi.

Le origini di QBasic

QBasic è un moderno insieme dell'originale linguaggio BASIC, creato da John G. Kemeny e Thomas E. Kurtz nel 1963 e 1964 al Dartmouth College, e di altre versioni di BASIC per i personal computer IBM e compatibili (come BASICA e GW-BASIC). QBasic non è soltanto un insieme di precedenti versioni del BASIC — in grado perciò di capirne le parole chiave e la sintassi — ma dispone di parole chiave e di funzionalità di altri famosi linguaggi per computer, il che lo rende un sistema completo per creare programmi. La Microsoft commercializza anche una versione più avanzata di Basic, detta Microsoft QuickBasic Compiler, di cui parleremo nel capitolo 13.

COME IMPARERÒ QBASIC CON QUESTO LIBRO?

Oltre a fornire un'introduzione sul linguaggio di programmazione QBasic, questo libro vi permette di mettere subito in pratica le conoscenze acquisite. Di pari passo con l'apprendimento del linguaggio scriverete programmi che mostrano dei caratteri sullo schermo, che leggono caratteri digitati sulla tastiera, che lavorano con parole e numeri e che conservano informazioni su un disco, recuperandole poi nuovamente. Imparerete anche a creare programmi che usano grafica e suono e, una volta completato il libro, sarete in grado di creare degli utili programmi con QBasic, il che vi permetterà di usare in modo ancor più efficiente il vostro computer.

NOTA: Ogni capitolo di questo libro (a partire dal prossimo) termina con domande ed esercizi di ripasso. Cercate di completarli dal momento che rappresentano una buona opportunità per rivedere quel che avete imparato e assicurarvi di aver capito ciò che è stato spiegato prima di proseguire. L'Appendice E contiene le risposte alle domande e i risultati degli esercizi.

QBASIC COME PARTE DI MS-DOS 5

QBasic viene distribuito gratuitamente con la versione 5 di MS-DOS o col pacchetto di aggiornamento. Se si dispone di questi prodotti (o di quelli compatibili distribuiti dai produttori di personal computer) si dispone anche di QBasic. Prima di cominciare, assicuratevi di aver installato la versione 5 di MS-DOS sul computer (se disponete di un disco fisso) o di avere dei dischetti (se disponete di un sistema con due unità a dischi flessibili e senza disco fisso). Il prossimo capitolo fornisce un'introduzione dettagliata per avviare QBasic e scrivere il vostro primo programma!

NOTA: Diamo per scontato che il lettore abbia una conoscenza almeno elementare del sistema operativo MS-DOS (formattazione dei dischi, cambiamento delle directory, impostazione di percorsi e così via). Se avete bisogno di rinfrescare alcuni concetti basilari, potete fare riferimento alla documentazione sul DOS che vi è stata fornita col vostro computer.

Appendice A: la vostra guida di riferimento a QBasic

Nei capitoli successivi si dà per scontato che il lettore abbia una qualche familiarità con ambienti basati sull'uso di menu, come MS-DOS EDIT (l'editore di testo della versione 5 di MS-DOS) o con Microsoft Works (un applicativo multifunzionale). Se avete bisogno di rivedere l'uso del mouse o della tastiera per orientarvi in un ambiente basato sull'uso di menu o se volete informazioni specifiche su come personalizzare l'ambiente QBasic, fate riferimento all'Appendice A, ''Uso dei menu e delle opzioni di QBasic'', che copre i seguenti argomenti:

- Selezione dei menu e dei comandi.
- Uso delle finestre di dialogo.
- Uso dei comandi di tastiera di QBasic.
- Selezionamento di testo.
- Guida in linea.
- Stampa.
- Modifica dei colori dello schermo.

CAPITOLO 2

Elementi base di QBasic

Dopo aver visto alcuni concetti introduttivi iniziamo subito a imparare a programmare.

AVVIAMENTO DI QBASIC

Per cominciare a programmare è necessario avviare QBasic; l'avviamento può avvenire da disco fisso o da disco flessibile a seconda del sistema disponibile.

NOTA: Nei successivi capitoli del libro il testo da digitare viene indicato in neretto.

Avviamento di QBasic da disco fisso

Se la versione 5 di MS-DOS è installata su disco fisso, il programma e i file di supporto di QBasic dovrebbero trovarsi nella directory DOS del drive C. Per avviare QBasic basta digitare i seguenti comandi e premere Invio:

```
c:
cd \dos
qbasic
```

NOTA: Per poter avviare QBasic da una qualsiasi directory sul disco fisso bisogna aggiungere la directory C: \DOS alla variabile PATH contenuta nel file AUTOEXEC.BAT.

Avviamento di QBasic da disco flessibile

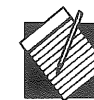
Se il vostro computer è dotato di due drive per dischi flessibili e non disponete di un disco fisso potete avviare QBasic nel modo seguente:

1. Inserite il disco di avviamento di MS-DOS 5 nel drive A.
2. Inserite il disco QBasic 1 nel drive B.
3. Riavviate il computer premendo Ctrl, Alt e Canc simultaneamente.
4. Al prompt del sistema digitate la seguente riga di comando PATH e premete Invio:

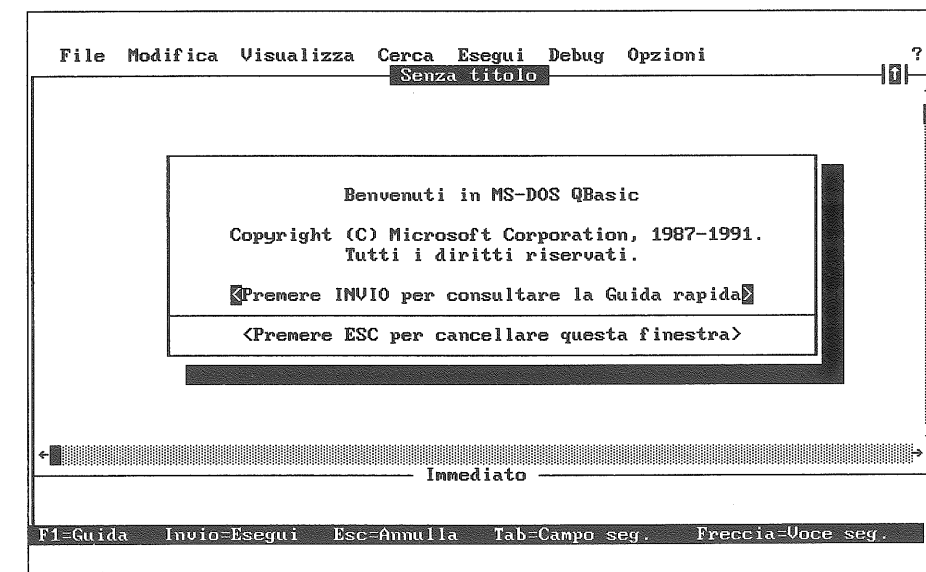
   ```
   path a:;b:
   ```


5. Avviate QBasic digitando la seguente riga di comando e premete Invio:

   ```
   qbasic
   ```

La luce del drive B si accenderà e QBasic apparirà sullo schermo. Quando la luce si spegne, rimuovete il disco di avviamento di MS-DOS 5 dal drive A e inseritevi il disco QBasic 2. Eseguite QBasic da entrambi i dischi fino a quando decidete di uscire dal programma: per far ciò rimuovete il disco QBasic 2 dal drive A e inseritevi il disco di avviamento di MS-DOS 5.

NOTA: Con la crescita nella dimensione dei programmi diventa sempre più difficile eseguirli da un sistema non dotato di disco fisso. Il maggior problema con QBasic sarà quello di esaurire facilmente lo spazio disponibile su disco e di dover cambiare dischetti molto spesso. Perciò, se usate il computer molto spesso può essere consigliabile comperare un disco fisso o un computer che ne sia dotato.

LO SCHERMO DI QBASIC

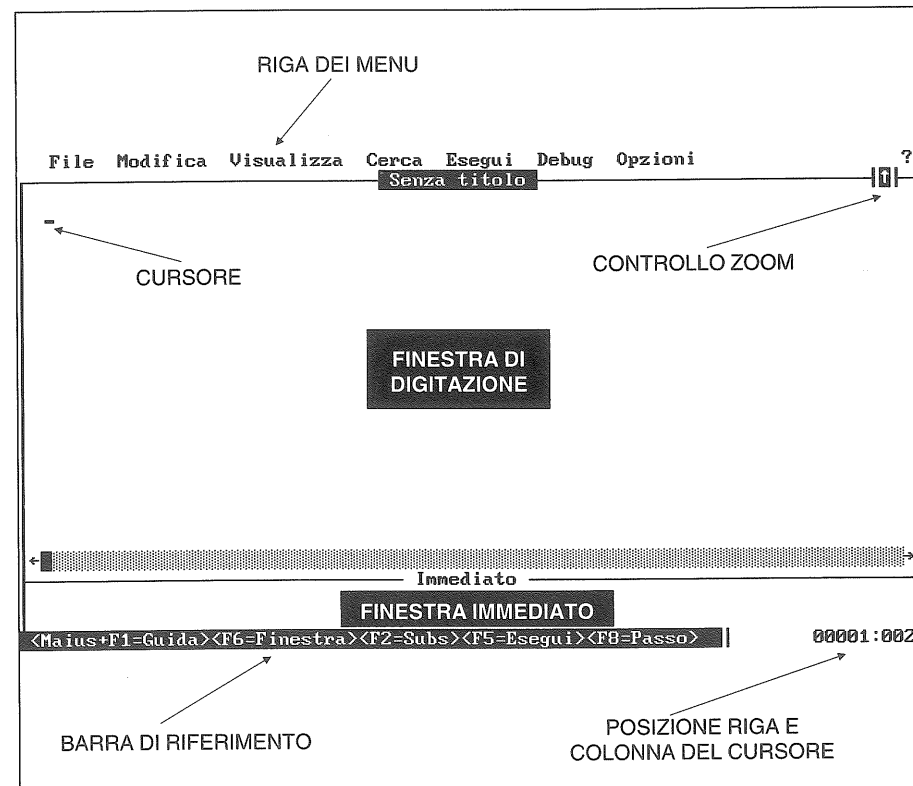
Lo schermo di QBasic, al momento dell'avviamento, appare nel modo seguente:

Questo messaggio di "benvenuto" appare ogni volta che QBasic viene avviato. Lo schermo offre due opzioni:

- Premendo Invio si può vedere il sistema di guida on line.
- Esc cancella il messaggio dallo schermo e avvia il programma.

Esercizio: Pulire lo schermo per cominciare a programmare

Dal momento che cominceremo a programmare già in questo capitolo, premete Esc per cancellare il messaggio dallo schermo.

L'ambiente QBasic

Cominciamo col descrivere i vari elementi di cui si compone lo schermo di QBasic, elementi che vi aiuteranno nella creazione dei programmi e nell'elaborazione dei testi.

Il cursore

Il trattino sottolineato lampeggiante che vedete nell'angolo superiore sinistro dello schermo è il cursore; esso indica il punto in cui apparirà il carattere digitato sullo schermo.

Il puntatore del mouse

Se avete installato un mouse sul vostro computer, un puntatore rettangolare appare nell'angolo superiore sinistro dello schermo; questo vi consente di spostarvi velocemente all'interno del programma e di selezionare facilmente i comandi dai menu.

Le coordinate di riga e colonna

Nell'angolo in basso a destra dello schermo ci sono due numeri separati da due punti. Questi valori indicano la posizione corrente del cursore: il primo fa riferimento alla riga in cui esso si trova, il secondo (quello dopo i due punti) alla colonna. Nella figura 2-1 i numeri indicano che il cursore si trova in corrispondenza della prima riga e della prima colonna.

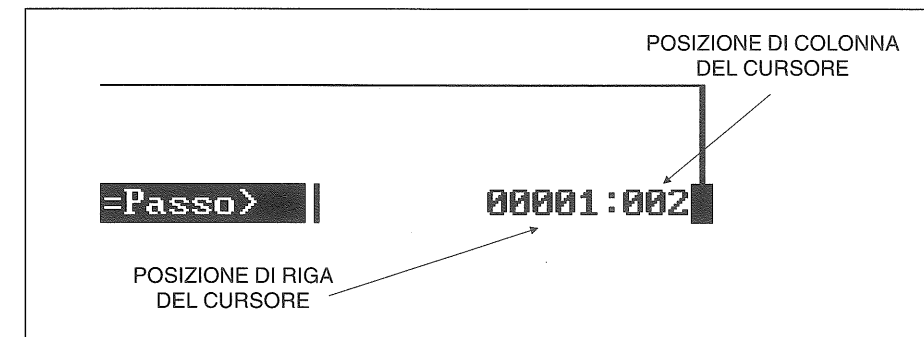



Figura 2-1. *Coordinate di riga e colonna.*

Esercizio: Cambiare le coordinate di colonna

1. Premete il tasto di direzione destro più volte e osservate come cambia la coordinata della colonna.

2. Premete il tasto di direzione sinistro finché il cursore torna nella posizione corrispondente alla prima colonna (più avanti, quando comincerete a scrivere dei programmi, avrete la possibilità di esercitarvi anche sul cambiamento delle coordinate di riga con i tasti di direzione Su e Giù).

La barra e i nomi dei menu

In cima allo schermo si può vedere la barra dei menu, che contiene i nomi di tutti i menu a scorrimento di QBasic.

La Finestra di digitazione e la Finestra Immediato

QBasic fornisce inizialmente due finestre nelle quali lavorare:

- La finestra superiore — in cui scrivere e lavorare sui programmi di propria creazione — è detta Finestra di digitazione.
- La finestra inferiore — in cui provare le istruzioni di programmazione prima di inserirle effettivamente nel programma — è detta Finestra Immediato.

È possibile lavorare, di volta in volta, in una sola di queste due finestre e il passaggio dall'una all'altra è molto facile. Viene detta finestra attiva quella in cui si sta lavorando in quel momento; essa può essere facilmente identificata:

- Osservando il cursore lampeggiante, che si trova sempre nella finestra attiva.
- Osservando il titolo che appare in cima alla finestra dal momento che quello della finestra attiva è evidenziato.

Si noti che il nome della Finestra di digitazione resta invariato (Senza titolo) fino a quando si attribuisce un nome al programma e lo si salva su disco.

La barra di riferimento

Alla base dello schermo si trova la barra di riferimento, che contiene una lista di tasti di controllo che si possono usare nella finestra attiva. Questa lista cambia ogni volta che si seleziona una nuova finestra attiva.

Esercizio: Cambiare le finestre con un comando della barra di riferimento.

Si noti che il tasto funzione F6 è il tasto Finestra, che viene usato per passare dall'una all'altra (se si dispone di un mouse basta invece posizionare il puntatore in un qualunque punto della finestra da attivare e fare clic). Provate a premere F6 e osservate ciò che succede.

Come potete vedere dalla figura 2-2, il cursore lampeggiante si trova ora nella Finestra Immediato. Le barre di scorrimento orizzontali e verticali ombreggiate sono scomparse dalla Finestra di digitazione, la parola Senza titolo

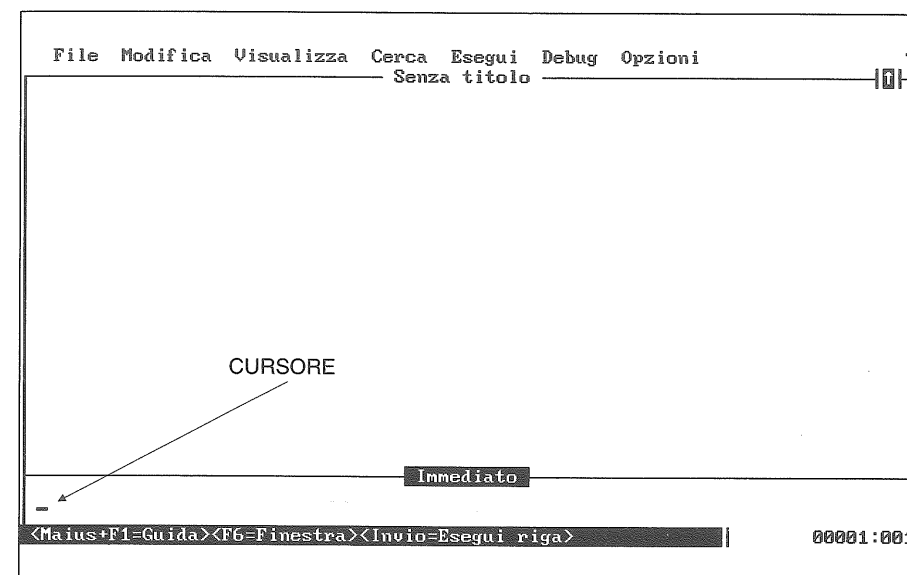



Figura 2-2. *Attivazione della Finestra Immediato.*

in cima a questa finestra non è più evidenziata, mentre lo è la parola Immediato in cima a questa finestra.

Questi cambiamenti indicano che la Finestra Immediato è diventata la finestra attiva; infatti, se provate a digitare dei caratteri sulla tastiera questi appariranno nella Finestra Immediato invece che nella Finestra di digitazione. Si noti che il titolo delle Finestra Immediato non cambia mai, a differenza di quello della Finestra di digitazione.

Si noti anche che nella barra di riferimento compare un diverso gruppo di tasti di controllo (quelli appunto della Finestra Immediato). Per il momento non useremo la Finestra Immediato e perciò potete premere F6 per rendere la Finestra di digitazione nuovamente attiva.

Il controllo zoom

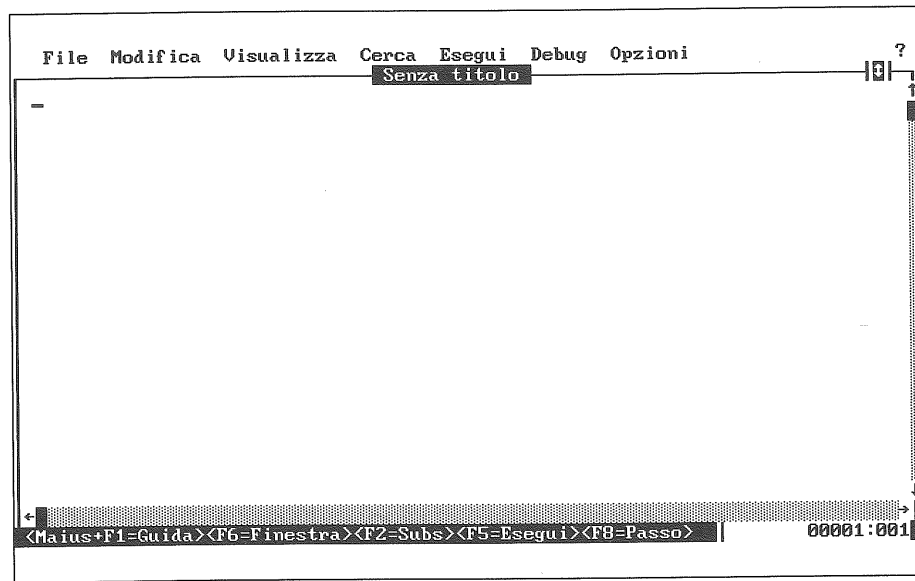
È possibile far scomparire dallo schermo la Finestra Immediato. Nell'angolo superiore destro dello schermo, appena sotto il menu Guida (?), si trova un rettangolo verticale contenente un freccia indirizzata verso l'alto detta controllo zoom. Questo controllo produce l'espansione della Finestra di digitazione che viene così a coprire la Finestra Immediato.

Una volta usato questo controllo, il carattere si trasforma in una freccia a due punte, per indicare che la Finestra di digitazione è stata espansa al massimo. Il controllo zoom può essere usato col mouse o con la tastiera:

- Se si usa il mouse basta fare clic nel rettangolo sotto il menu Guida (?). Per riportare la Finestra di digitazione alla dimensione originale basta fare clic nuovamente nella stessa area.
- Se si usa la tastiera bisogna tenere premuto il tasto Ctrl e premere il tasto F10. Per riportare la Finestra di digitazione alla dimensione originale basta premere di nuovo Ctrl+F10.

Esercizio: Espansione della Finestra di digitazione con il controllo zoom

1. Usate il mouse o la tastiera per espandere la Finestra di digitazione con il controllo zoom; lo schermo apparirà in questo modo:

```
File  Modifica  Visualizza  Cerca  Esegui  Debug  Opzioni                  ?
                        Senza titolo
<Maius+F1=Guida><F6=Finestra><F2=Subs><F5=Esegui><F8=Passo>      00001:001
```

2. Ora riportate la Finestra di digitazione alla sua dimensione originale per rendere nuovamente visibile la Finestra Immediato.

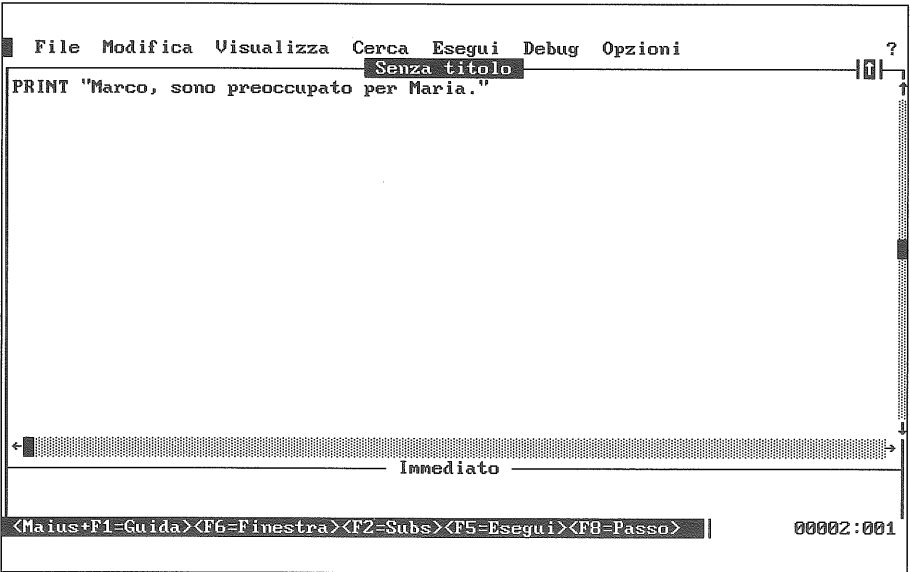
IL VOSTRO PRIMO PROGRAMMA

Dopo aver brevemente introdotto QBasic cercheremo ora di scrivere un primo, semplice programma in QBasic. Scrivere questo programma è molto semplice: basterà digitare le righe esattamente come viene mostrato negli esempi seguenti e premere Invio alla fine di ogni riga.

Esercizio: Scrivere il vostro primo programma

1. Portate il cursore in corrispondenza della prima riga e della prima colonna e digitate la riga seguente, esattamente come appare sotto — spazi, virgolette e così via:

```
print "Marco, sono preoccupato per Maria."
```

2. Controllate la riga attentamente e assicuratevi di non aver commesso errori di battitura. In questo caso usate il tasto Backspace per correggere l'errore e poi completate la riga come nell'esempio.
3. Quando siete certi che non ci sia alcun errore premete Invio; a questo punto lo schermo apparirà nel modo seguente:

Si noti che la parola PRINT è in maiuscolo perché essa rappresenta una parola chiave di QBasic — fa parte cioè del "vocabolario" di QBasic. Le parole chiave vengono sempre trasformate in maiuscole per distinguerle più facilmente dalle altre.

La riga che abbiamo appena digitato, per quanto breve, costituisce un completo programma in QBasic.

Eseguire un programma

Per eseguire un programma si usa il comando Avvia dal menu Esegui; questo comando dice a QBasic di eseguire le istruzioni contenute nella Finestra di digitazione.

Esercizio: Eseguire il vostro primo programma

Per eseguire il programma appena scritto premete Alt per attivare la barra dei menu, scegliete Esegui e poi il comando Avvia (per una rassegna sull'uso della tastiera o del mouse nella selezione dei comandi di menu si veda l'Appendice A, "Uso dei menu e delle opzioni di QBasic"). A questo punto lo schermo dovrebbe apparire così:

```
C:\ <Mer 30/10/1991> >qbasic
Marco, sono preoccupato per Maria.







Premere un tasto per continuare
```


Lo schermo di output

QBasic mostra l'output del programma, cioè il risultato finale delle istruzioni digitate, nello *schermo di "output"*. Questo schermo viene attivato automaticamente quando si esegue un programma e ciò per evitare che il risultato del programma sovrascriva le istruzioni (dette anche codice) contenute nella Finestra di digitazione. Lo schermo di output mostra qualsiasi programma eseguito in precedenza, a meno che venga ripulito. Questa è la ragione per cui, nell'illustrazione precedente, potete vedere la riga di comando di DOS usata per eseguire QBasic. Il messaggio alla base dello schermo di output, "Premere un tasto per continuare", compare quando l'esecuzione del programma è terminata.

Esercizio: Rattivare la Finestra di digitazione

Premete un tasto qualsiasi per riattivare la Finestra di digitazione. Congratulazioni! Avete appena scritto ed eseguito il vostro primo programma!

Cambiare un programma

Abbiamo appena visto una delle parole chiave di QBasic, PRINT, la cui funzione è di visualizzare sullo schermo le informazioni inserite nel programma. Vedremo ora un'altra parola chiave, detta CLS (CLear Screen, Pulisci Schermo), che permette di cancellare completamente il contenuto dello schermo di output. Proveremo ora ad aggiungere questa istruzione al programma appena scritto, per vedere come modificare un programma esistente.

Organizzare in modo ordinato le istruzioni di QBasic

Prima di aggiungere CLS al programma bisogna riflettere sull'ordine in cui si vuole che queste vengano eseguite dal programma. Si tenga presente che negli esempi di questo libro ogni istruzione si trova su una riga diversa e che, quando si esegue un programma, QBasic comincia dalla prima riga ed esegue le istruzioni una ad una, nell'ordine in cui sono state inserite, fino al termine della lista. Il primo passo è dunque quello di decidere dove inserire l'istruzione CLS nel programma.

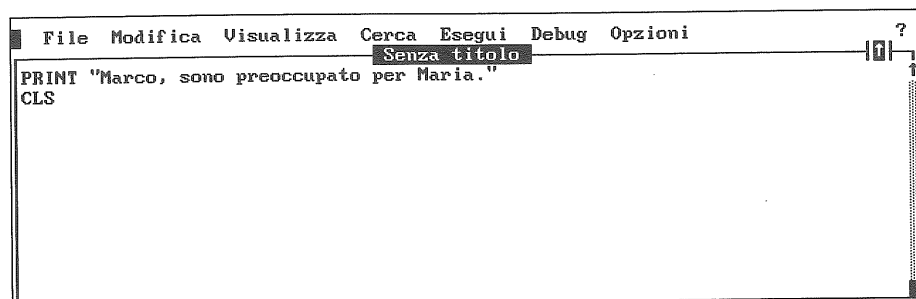
Esercizio: Aggiungere un'istruzione al programma

In questo momento il cursore si trova in corrispondenza alla lettera P dell'istruzione PRINT.

1. Digitate la riga seguente e premete Invio.

   ```
   CLS
   ```

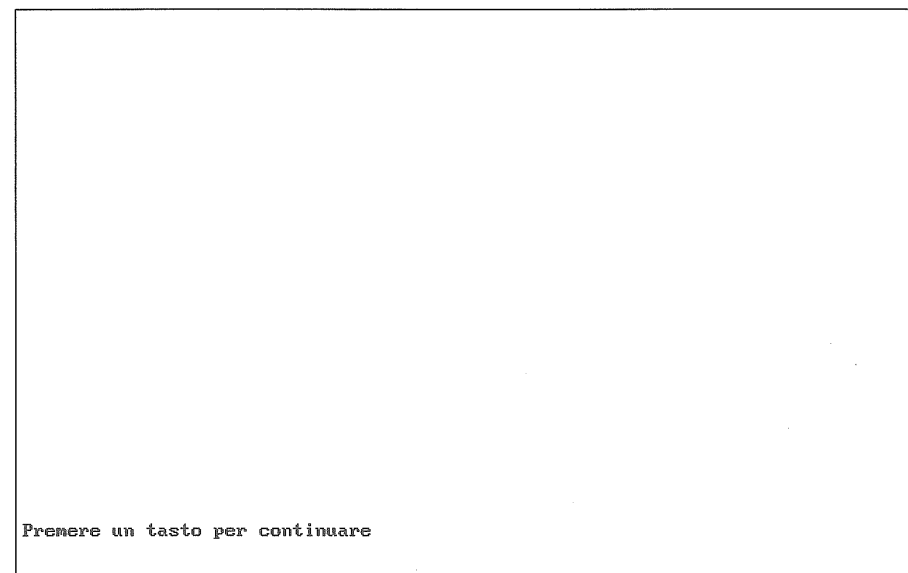
   Lo schermo dovrebbe ora apparire in questo modo:

   ```
   File  Modifica  Visualizza  Cerca  Esegui  Debug  Opzioni            ?
                                 Senza titolo
   PRINT "Marco, sono preoccupato per Maria."
   CLS
   ```

   (Come visto nell'esempio precedente QBasic trasforma l'istruzione CLS in lettere maiuscole in quanto si tratta di una parola chiave.)

2. Eseguite di nuovo il programma: scegliete Avvia dal menu Esegui. Se fate attenzione potrete vedere il messaggio "Marco, sono preoccupato per Maria." lampeggiare brevemente sullo schermo e poi scomparire per lasciare lo schermo di output come appare nella figura successiva. Non ha perciò molto senso usare un'istruzione PRINT se questa viene immediatamente seguita da CLS; questo ci fa capire quanto importante sia stabilire un corretto ordine per le istruzioni contenute nel programma. L'istruzione CLS va posta prima di PRINT se si vuole che il programma si comporti nel modo desiderato.

3. Premete un tasto qualsiasi per riattivare la Finestra di digitazione.

Modificare un programma

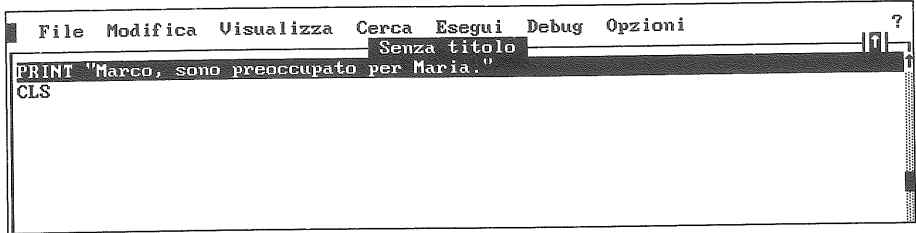
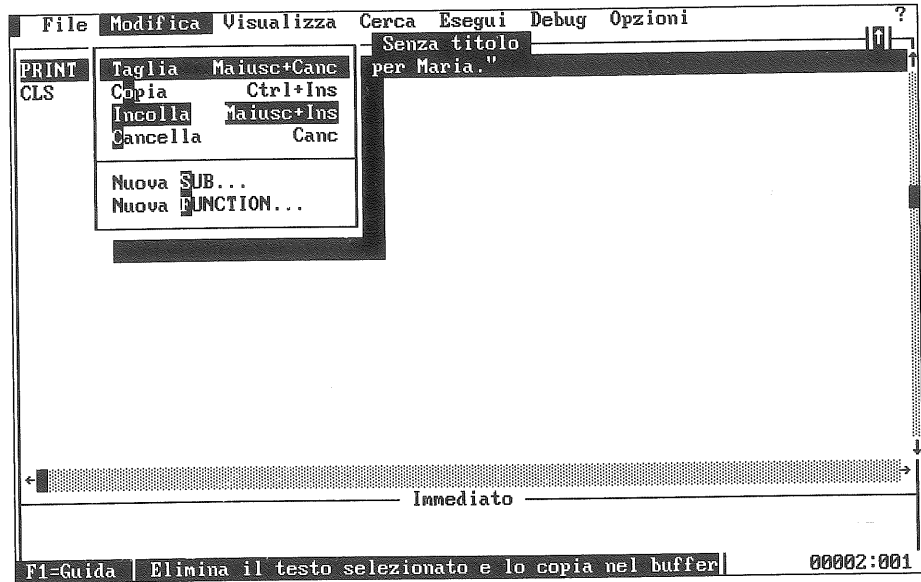
Il processo di cambiamento di un programma viene anche detto di modifica. Modificare un programma implica aggiungere o cancellare caratteri e spostare righe di istruzioni da una parte all'altra.
QBasic fornisce diversi strumenti di modifica: alcuni di questi si trovano all'interno del menu Modifica; altri sono rappresentati da tasti o loro combinazioni. Le esercitazioni seguenti rappresentano un'utile rassegna generale di

questi strumenti, ma se volete informazioni più dettagliate potete fare riferimento all'Appendice A, "Uso dei menu e delle opzioni di QBasic".

I comandi Taglia, Copia e Incolla

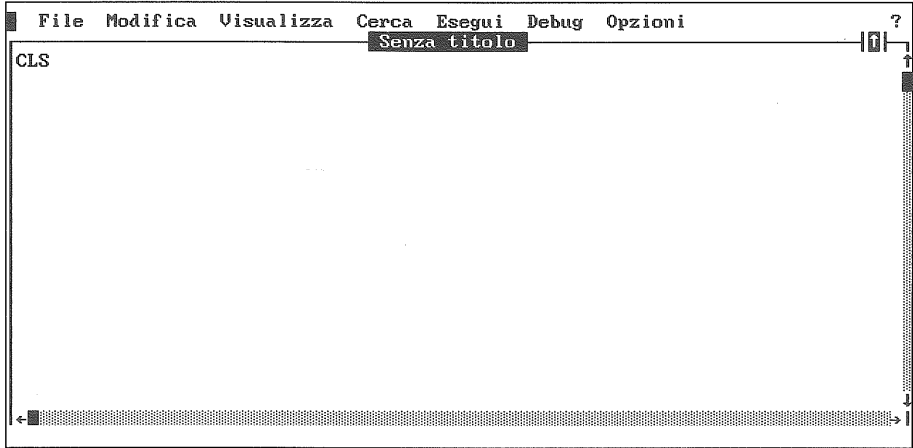
I comandi Taglia, Copia e Incolla rappresentano un efficiente insieme di strumenti per la modifica dei programmi. Consentono infatti di risparmiare tempo e di ridurre le possibilità di commettere errori di battitura.

Esercizio: Modifica di un programma coi comandi Taglia e Incolla

Abbiamo visto nell'esempio precedente che l'istruzione PRINT dev'essere spostata dopo l'istruzione CLS. Per far ciò si usano i comandi Taglia e Incolla del menu Modifica.

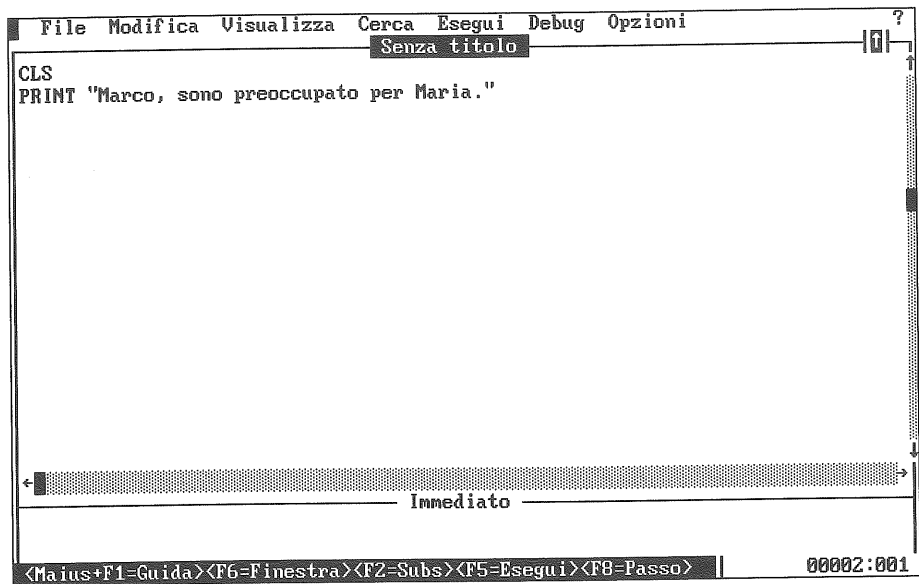
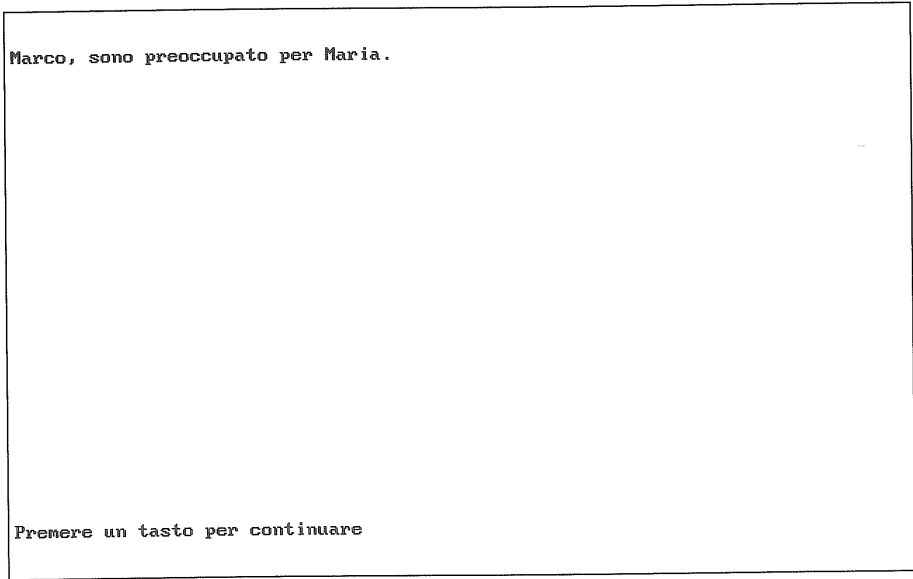
1. Spostate il cursore sotto la lettera P dell'istruzione PRINT; tenete premuto il tasto Maiusc e premete il tasto di direzione Giù per selezionare l'intera riga.
   Lo schermo dovrebbe apparire come nella prima figura di pagina seguente.

2. Aprite il menu Modifica:

3. Scegliete il comando Taglia dal menu Modifica; potrete vedere che l'istruzione PRINT scompare dalla Finestra di digitazione e resta solo l'istruzione CLS (come mostrato nella figura successiva). Sebbene non sia visibile sullo schermo, l'istruzione PRINT non è scomparsa definitivamente, ma è stata posta negli Appunti di QBasic.
   Il passo successivo sarà perciò quello di incollarla (Incolla) dagli Appunti nuovamente all'interno del programma.

Gli Appunti

Gli Appunti di QBasic conservano il testo cancellato o copiato per ultimo dal programma. Questo testo resta negli Appunti fino a quando dell'altro testo viene cancellato o copiato o fino all'uscita da QBasic. Non è possibile cancellare il contenuto degli Appunti.

4. Spostate il cursore nel punto in cui desiderate inserire il testo. Per inserire l'istruzione PRINT dopo CLS basta premere il tasto di direzione Giù una volta per collocare il cursore sotto la riga che contiene l'istruzione CLS.

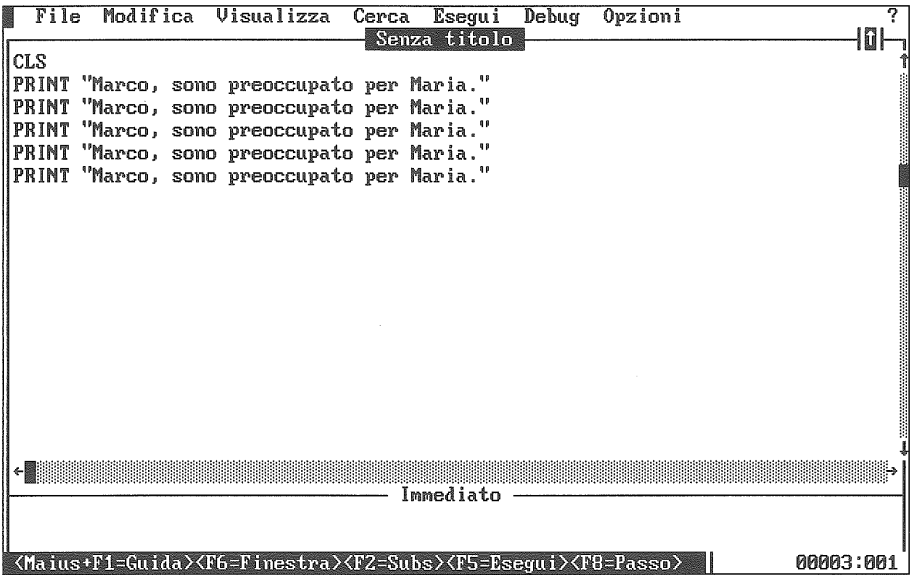
5. Aprite il menu Modifica.

6. Scegliete il comando Incolla. Lo schermo dovrebbe ora apparire come nella figura di pagina seguente.

NOTA: Se, dopo aver completato quest'operazione, l'istruzione PRINT viene inserita prima di CLS, è probabile che non abbiate collocato correttamente il cursore. Ricordate che se volete incollare il contenuto degli Appunti dopo una particolare istruzione, dovete assicurarvi che il cursore si trovi nella riga immediatamente sotto questa, prima di eseguire il comando Incolla. Ripetete quest'operazione, se necessario, in modo che l'istruzione PRINT si venga a trovare dopo CLS.

7. Provate a eseguire il programma per vedere il risultato del cambiamento:

8. Premete un tasto qualsiasi per riattivare la Finestra di digitazione

Esercizio: Modifica di un programma coi comandi Copia e Incolla

Dopo aver posizionato correttamente l'istruzione CLS, proviamo ora ad ampliare il nostro programma usando i comandi Copia e Incolla.

1. Selezionate nuovamente l'intera istruzione PRINT.
2. Scegliete il comando Copia dal menu Modifica; notate che la riga selezionata non scompare dal programma.
3. Spostate il cursore una riga sotto quella dell'istruzione PRINT e potete notare che questa non è più evidenziata.
4. Scegliete Incolla dal menu Modifica
5. Scegliete Incolla di nuovo, fino all'inserimento di cinque istruzioni PRINT. Il vostro programma dovrebbe ora apparire in questo modo:

Se eseguite ancora il programma vedrete lo stesso messaggio comparire più volte sullo schermo. Vediamo di imparare come si cancellano alcune di queste istruzioni.

Esercizio: Cancellare testo col tasto Canc

1. Spostate il cursore in corrispondenza alla lettera P dell'ultima istruzione PRINT e premete il tasto Canc più volte; potrete notare che questo tasto cancella il carattere sottolineato di volta in volta dal cursore.

2. Continuate a premere Canc fino a che l'intera riga è stata cancellata.

3. Se premete Canc ancora, alla fine comincerete a sentire un suono; ciò sta a indicare che non ci sono più caratteri da cancellare in quella riga (in generale, ogni volta che sentite un suono di questo genere significa che state cercando di fare qualcosa che QBasic non ammette).

Esercizio: Cancellare testo col tasto Backspace

1. Spostate il cursore su di una riga.

2. Premete il tasto Fine per spostare il cursore alla fine della riga.

3. Premete Backspace ripetutamente fino a far scomparire l'intera istruzione. Notate che se continuate a premere Backspace una volta cancellata l'intera riga, il cursore si sposta alla fine della riga superiore e comincia a cancellare i caratteri in essa contenuti.

Esercizio: Cancellare testo selezionato col tasto Canc

1. Selezionate una parte o tutta l'ultima istruzione PRINT.

2. Premete Canc per far scomparire definitivamente tutto il testo selezionato (se ne avevate selezionato solo una parte, cancellate anche la rimanente). Premendo il tasto Canc al posto del comando Taglia si dice a QBasic di non conservare il testo cancellato negli Appunti.

Esercizio: Cancellare un'intera riga con Ctrl-Y

1. Spostate il cursore in un qualsiasi posto all'interno di una riga dell'istruzione PRINT.

2. Tenete premuto Ctrl e premete Y; otterrete lo stesso risultato che avreste ottenuto selezionando l'intera riga e usando il comando Taglia.

Ecco eliminate tutte le istruzioni PRINT! Infatti, se avete seguito questi esercizi dovreste ora trovarvi con l'istruzione CLS e una sola istruzione PRINT.

Modifica del testo

A volte si potrebbe voler cambiare solo parte di un'istruzione. Per far ciò la si può cancellare interamente e poi riscriverla, oppure si può modificare soltanto la parte interessata; questa seconda soluzione consente di risparmiare tempo e di ridurre i rischi di errore.

Prendiamo in considerazione un'istruzione PRINT, la quale visualizza (stampa) qualunque testo si trovi tra le virgolette che la seguono. Se avete usato questa istruzione in un vostro programma e decidete di cambiarne il messaggio, tutto ciò che dovete fare è modificare il testo tra le virgolette; per far ciò potete seguire diversi metodi.

Potete cancellare il testo esistente (usando uno dei metodi descritti in precedenza) e digitare il nuovo messaggio; oppure potete semplicemente digitare il nuovo testo usando uno dei due cursori disponibili: il cursore di inserimento e il cursore di sovrascrittura.

Il cursore di inserimento

Il cursore usato fino a questo momento (rappresentato da un segno di sottolineatura lampeggiante) è detto cursore di inserimento, perché consente di inserire del testo in mezzo a una riga senza eliminare quello già esistente. Per inserire testo usando il cursore di inserimento basta seguire questi passi:

1. Spostate il cursore nel punto in cui volete inserire il testo.

2. Digitate il testo desiderato.

Esercizio: Inserimento di testo col cursore di inserimento

1. Spostate il cursore in corrispondenza della prima lettera dopo le virgolette di apertura del messaggio dell'istruzione PRINT.

2. Digitate alcune lettere e osservate quel che succede: il messaggio esistente si sposta verso destra lasciando spazio al nuovo testo digitato.

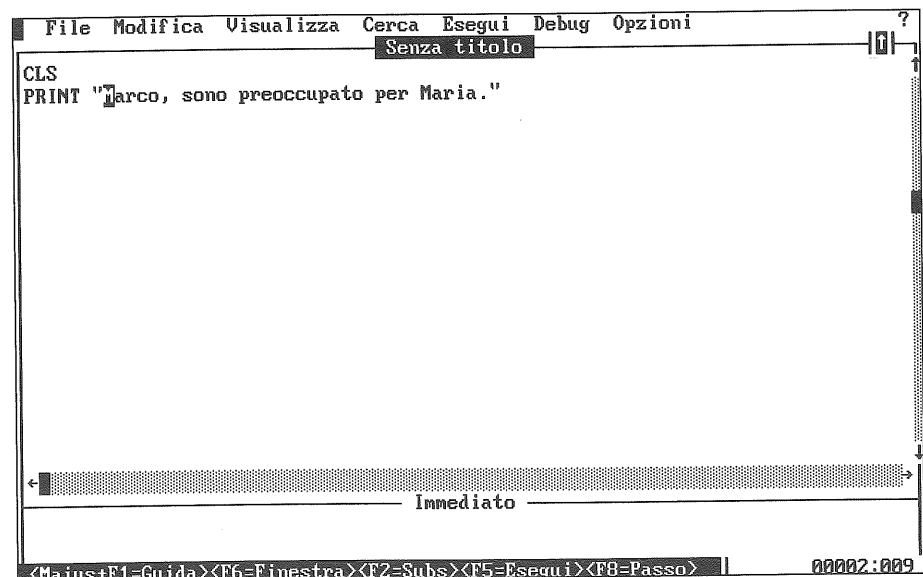
3. Usate il tasto Backspace per cancellare le lettere inserite e ristabilire il messaggio originale.

Il cursore di sovrascrittura

L'altro cursore viene detto di sovrascrittura, perché sostituisce i caratteri esistenti con quelli digitati. L'uso di questo cursore può talvolta consentire un risparmio di tempo, dal momento che non si deve cancellare alcun carattere. Per usare il cursore di sovrascrittura si possono seguire questi passi:

1. Spostate il cursore all'inizio dei caratteri da sovrascrivere.
2. Premete il tasto Ins; il cursore cambia forma diventando un rettangolo lampeggiante.
3. Digitate i caratteri desiderati.

Esercizio: Uso del cursore di sovrascrittura

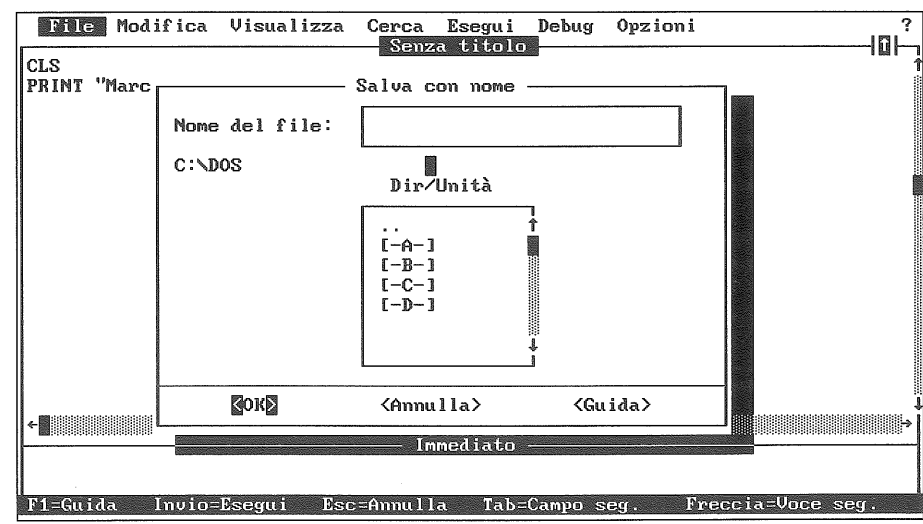
1. Spostate il cursore in corrispondenza della prima lettera dopo le virgolette di apertura del messaggio dell'istruzione PRINT.
2. Premete il tasto Ins; il cursore cambia forma diventando un rettangolo lampeggiante.
3. Iniziate a digitare il nuovo messaggio e osservate cosa succede: i nuovi caratteri sostituiscono quelli precedenti.

4. Se il nuovo messaggio è più breve di quello precedente potete usare Canc per eliminare i caratteri rimanenti (ricordatevi di non cancellare le virgolette di chiusura); se invece è più lungo ricordatevi di inserire delle virgolette di chiusura alla fine del messaggio.
5. Eseguite il programma per vedere il nuovo messaggio e poi ritornate alla Finestra di digitazione, premendo un tasto qualsiasi.
6. Premete Ins per riportare il cursore in modalità inserimento. Vi suggeriamo di mantenere il cursore in questa modalità nel prosieguo del libro.

Salvataggio del programma su disco

Dopo aver scritto un programma, se volete conservarlo, dovete salvarlo su disco. Se non lo salvate, il programma andrà perso quando il computer viene spento e sarete costretti a riscriverlo da capo.

Il programma appena scritto è molto breve e non richiederebbe un grande sforzo se dovesse essere riscritto; ma quando si comincia a scrivere programmi di una certa dimensione diventa assolutamente consigliabile salvarli su disco.

Esercizio: Salvataggio di un programma su disco

1. Scegliete il comando Salva con nome... dal menu File, che visualizza la seguente finestra di dialogo:

2. Digitate questo testo nella casella Nome del file:

```
primo.bas
```

NOTA: Siccome QBasic parte dal presupposto che il file che viene salvato sia un programma, gli attribuisce automaticamente l'estensione .BAS anche senza che questa venga digitata.

3. Premete Invio per eseguire il comando Salva con nome...; QBasic salverà il vostro programma col nome PRIMO.BAS.

Dopo aver salvato il programma su disco, QBasic riattiva la Finestra di digitazione e il nome del file compare in cima alla finestra; in questo modo è sempre possibile sapere su quale programma si sta lavorando e, se il titolo della finestra è "Senza titolo", si può dedurre che il programma non è ancora stato salvato.

Non è necessario che un programma sia completo prima che possa essere salvato. Come si fa a decidere quando un programma dev'essere salvato per la prima volta e quanto spesso dev'essere salvato dopo quella prima volta? Per rispondere a questa domanda basta che vi chiediate quanto lavoro andrebbe perso se venisse accidentalmente a mancare la corrente.

Attribuzione del nome ai file

I suggerimenti che seguono vi possono aiutare nel decidere come nominare correttamente i vostri file:

- Un nome di file non può essere più lungo di otto caratteri.
- Più un nome è descrittivo del contenuto del file meglio è.
- Un nome di file può essere una combinazione di lettere, numeri e alcuni simboli di interpunzione; i caratteri seguenti e gli spazi non possono essere usati:

  * = + {} ; : ' ? / \ |

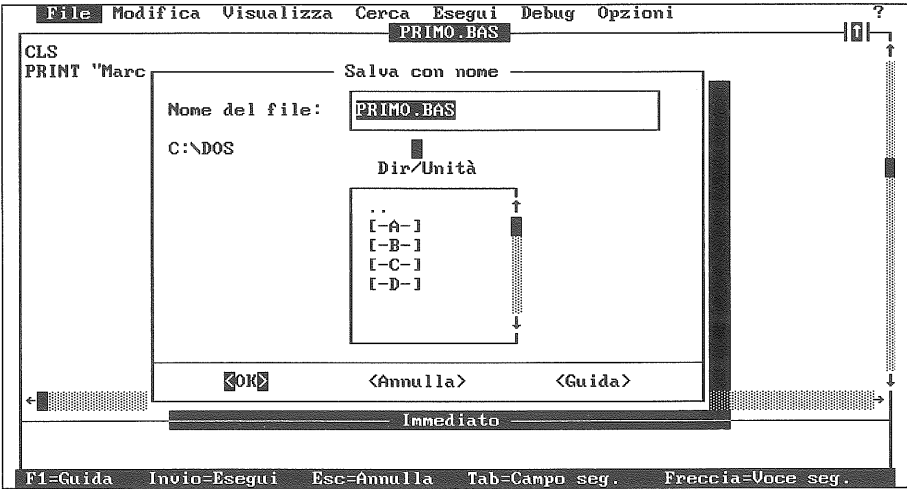
L'estensione di un file di programma dovrebbe sempre essere .BAS, perché QBasic riconosce questa estensione automaticamente.

Cambiamento di un file salvato

Se si apportano cambiamenti a un programma dopo che è stato salvato, il programma modificato viene trattato da QBasic come un programma separato all'interno della Finestra di digitazione. Ciò significa che QBasic non vi permetterà di lavorare su un altro programma prima di sapere che cosa volete fare con la versione modificata del programma corrente: volete sovrascrivere la versione salvata con quella modificata oppure volete che i cambiamenti vengano scartati?

Esercizio: Lavorare con cambiamenti a un file salvato

1. Apportate dei cambiamenti al programma nella Finestra di digitazione (potete fare i cambiamenti che volete, per esempio cambiando il messaggio dell'istruzione PRINT o aggiungendo una seconda istruzione).

2. Dopo aver effettuato i cambiamenti scegliete il comando Salva con nome... dal menu File. Si noti che la casella Nome del file contiene il nome che era stato dato al programma in precedenza; a questo punto avete più opzioni:

- ☐ Scegliere OK per salvare la versione modificata del programma, sostituendo quella precedente.

- Digitare un nuovo nome per il programma modificato e scegliere OK. In questo modo si salva la versione modificata sotto un nuovo nome, mantenendo inalterata quella originale.
- Scegliere Annulla per cancellare l'operazione di salvataggio e ripristinare la Finestra di digitazione.
- Scegliere Guida (?) per visualizzare un'altra finestra di dialogo che spiega il contenuto della finestra del comando Salva con nome....

3. Premete Invio per scegliere OK e salvate il programma modificato al posto di quello originale.

Salvare spesso

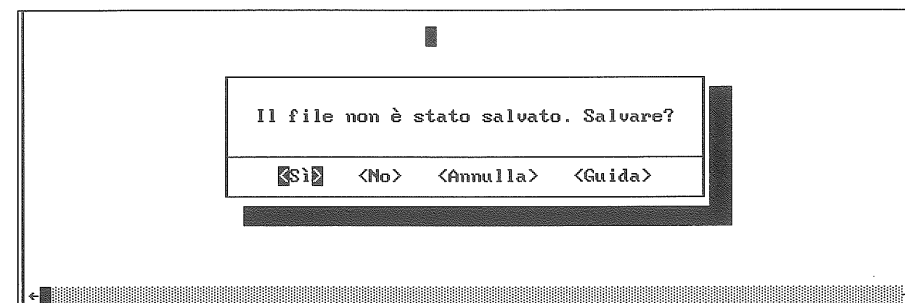
Fino a quando il programma non viene salvato su disco, tutto ciò che viene digitato nella Finestra di digitazione viene conservato nella memoria temporanea del computer. Cercate di salvare i vostri programmi su disco con regolarità per evitare di perdere informazioni; basta infatti che si verifichi una temporanea perdita di corrente perché tutte le informazioni conservate nella memoria temporanea del computer vadano perse. Se invece conservate il vostro programma su disco (fisso o flessibile) potrete sempre recuperarlo con facilità.

INIZIARE UN NUOVO PROGRAMMA

La Finestra di digitazione può contenere un solo programma alla volta (sia che si tratti di un programma appena digitato sia che si tratti di un programma caricato da disco). Per ripulire la Finestra di digitazione dal programma corrente e cominciare a scriverne uno nuovo basta scegliere il comando Nuovo dal menu File. Se il programma nella Finestra di digitazione è stato modificato QBasic non vi permetterà di avviarne uno nuovo fino a che non deciderete che cosa fare della versione modificata.

Esercizio: Avviamento di un nuovo programma

1. Apportate qualche modifica al programma contenuto nella Finestra di digitazione (per esempio modificate il messaggio dell'istruzione PRINT).

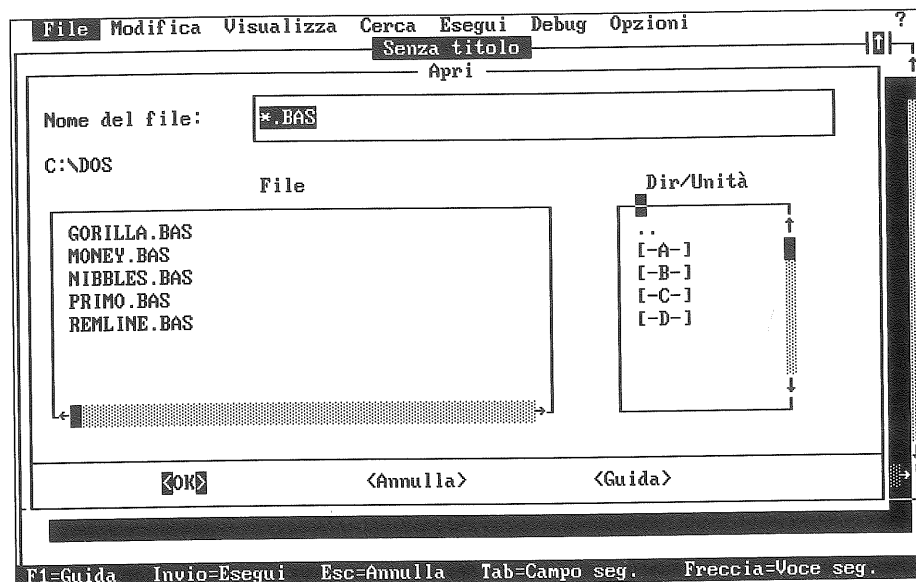
2. Dopo aver effettuato il cambiamento scegliete Nuovo dal menu File e vedrete apparire questa finestra di dialogo:

Se premete Invio scegliendo la risposta predefinita, "Sì", QBasic salva la versione modificata del programma usando il nome che appare in cima alla Finestra di digitazione e sovrascrive la versione precedente. In questo caso non è possibile digitare un nome diverso (per far ciò bisogna premere Esc per uscire da questa finestra di dialogo e poi scegliere il comando Salva con nome... dal menu Modifica e attribuire al programma modificato un nuovo nome).
Se scegliete il pulsante "No" QBasic non salva i cambiamenti apportati dall'ultima volta che il file era stato salvato; fate bene attenzione all'uso di questo pulsante, perché rischiate di perdere importanti modifiche.

3. Se volete salvare il programma modificato e sovrascrivere la versione precedente, premete Invio; altrimenti premete il tasto Tab per passare al pulsante "No" e poi premete Invio.
QBasic pulisce la Finestra di digitazione permettendo di scrivere un nuovo programma. Provate a scrivere alcune istruzioni PRINT per esercitarvi su quanto abbiamo visto fino ad ora e poi salvate il programma.

APERTURA DI UN PROGRAMMA ESISTENTE

Per lavorare con un programma conservato su disco, bisogna prima aprirlo, così come bisogna aprire un libro per poterlo leggere; quando si apre un programma questo viene caricato nella Finestra di digitazione. Per aprire un programma si usa il comando Apri... del menu File.

Esercizio: Apertura di un programma esistente

1. Aprite il menu File e osservate che il comando Apri... è seguito da dei puntini di sospensione (...).
2. Scegliete il comando Apri... per visualizzarne la finestra di dialogo.

3. Premete Tab per spostare il cursore nella casella Nome del file.
4. Evidenziate REVERSI.BAS.
5. Scegliete OK e QBasic apre il programma caricandolo nella Finestra di digitazione.

Ora avete la possibilità di modificare o eseguire il programma REVERSI.BAS (un gioco). Questo metodo viene usato per caricare qualsiasi file esistente su disco (si può usare la casella Dir/Unità per individuare file collocati in altre aree del disco).

USCITA DA QBASIC

Se avete terminato la vostra sessione di programmazione è consigliabile uscire da QBasic prima di spegnere il computer; in questo modo eviterete di dimenticarvi di salvare qualcosa di importante, che andrebbe automaticamente persa allo spegnimento del computer.

Per uscire da QBasic scegliete Esci dal menu File; ricordate che se la Finestra di digitazione contiene un programma che non è stato salvato o che è stato modificato QBasic vi chiederà se intendete salvarlo oppure no.

SOMMARIO

In questo capitolo avete potuto vedere alcuni elementi importanti per imparare a usare QBasic. Avete appreso come avviare, usare e uscire da QBasic, come scrivere, eseguire, modificare e salvare i vostri programmi. Nel prossimo capitolo vedremo, in modo più approfondito, un gruppo di nuove istruzioni.

DOMANDE ED ESERCIZI

1. Come si avvia QBasic?
2. Vero o Falso: l'apertura di un file di programma conservato su disco cancella quel file dal disco stesso.
3. Qual è la differenza tra la Finestra di digitazione e la Finestra Immediato?
4. Qual è la differenza, nel menu File, tra il comando Nuovo e Apri...?
5. Qual è la differenza tra la Finestra di digitazione e lo schermo di output?
6. Qual è la differenza, nel menu Modifica, tra il comando Taglia e Copia?
7. Quali sono i quattro modi per cancellare testo da un programma e in che cosa differiscono l'uno dall'altro?
8. Qual è la differenza tra il cursore di inserimento e quello di sovrascrittura? Come posso individuare quale dei due è attivo? Come posso passare dall'uno all'altro?
9. Escludendo l'estensione .BAS qual è il massimo numero di caratteri che si possono usare per nominare un file?
10. Quali dei nomi seguenti non può essere il nome di un programma? Perché no?

```
NOMESTAMPA.BAS;        SALVE!.BAS;          BLU+BLU.BAS;
BLU BLU. BAS;          ABC-E-DEF.BAS;       BUSINESS.BAS;
PALLONE.BAS;           PROG[1].BAS;         50%FATTO.BAS;
1*PROG.BAS;            MODO_MIO.BAS;        VA-BENE.BAS
```

11. Come si esce da QBasic? Che cosa succede se si cerca di uscire prima di aver salvato il lavoro compiuto e quali opzioni si presentano?

CAPITOLO 3

Introduzione al linguaggio QBasic

Nell'ultimo capitolo abbiamo potuto acquisire una certa familiarità con QBasic e abbiamo usato un paio di istruzioni per scrivere un semplice programma. In questo capitolo esamineremo più in profondità il linguaggio QBasic, ampliando alcuni concetti già espressi in precedenza.

ANATOMIA DI UN'ISTRUZIONE QBASIC

Fino ad ora abbiamo fatto riferimento a una riga di QBasic come a un'istruzione. Come avete imparato nel capitolo 1 ogni riga di un programma QBasic è semplicemente un'istruzione che il computer esegue quando si fa girare il programma. QBasic riconosce due tipi di istruzioni: *enunciati* e *funzioni.*

Enunciati e funzioni

Gli enunciati e le funzioni di QBasic sono molto simili e ugualmente facili da usare; la differenza principale è questa:

- Un enunciato è un'istruzione diretta che fa semplicemente ciò che le viene detto quando si esegue il programma. Quando il programma esegue un enunciato il risultato è, nella maggior parte dei casi, evidente: l'enunciato PRINT, per esempio, mostra dei caratteri sullo schermo output e l'enunciato CLS lo pulisce.
- Una funzione restituisce un valore che il programma può usare; il suo funzionamento è meno ovvio di quello di un enunciato. Essa generalmente compare all'interno di un enunciato e svolge il suo compito al momento dell'esecuzione dell'enunciato.

Enunciati e funzioni hanno una propria *sintassi,* che serve a regolare il modo in cui vengono usati nel programma.

La sintassi di QBasic

La sintassi di un'istruzione QBasic, che si tratti di una funzione o di un enunciato, è costituita semplicemente da una parola chiave seguita dalle informazioni necessarie all'esecuzione del compito.

La sintassi di alcune istruzioni QBasic (come BEEP) consiste solamente nel nome dell'enunciato o della funzione. Avviate QBasic e digitate il seguente programma costituito da una sola riga:


```
BEEP
```

Eseguite il programma e osservate che cosa succede: l'enunciato BEEP fa sì che l'altoparlante interno al computer emetta un breve suono.

Ora cercheremo di dare un'occhiata a istruzioni più complesse; nel corso di questo capitolo useremo l'enunciato PRINT come base per i nostri esempi.

Sintassi dell'enunciato PRINT

L'enunciato PRINT presenta questa sintassi:

```
PRINT [listaespressione] [,|;]
```

Come potete vedere non è molto simile all'enunciato PRINT visto nel secondo capitolo! PRINT è infatti un enunciato molto versatile e nel capitolo precedente abbiamo usato solo una delle sue molte capacità. Diamo un'occhiata più da vicino alla struttura di questa riga di sintassi:

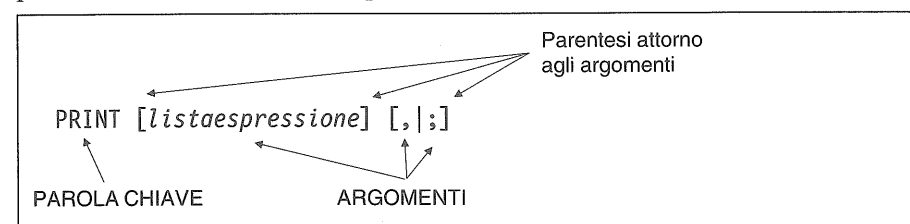



Figura 3-1. *Struttura di una riga di sintassi.*

Con l'eccezione della parola chiave ogni elemento di una riga di sintassi viene detto *argomento*; esso fornisce all'istruzione le informazioni addizionali necessarie per l'esecuzione del suo compito. Alcuni argomenti sono facoltativi; essi vengono mostrati tra parentesi quadre sia in questo libro sia nella guida on line di QBasic, ma se decidete di usarli in una vostra istruzione *non dovete racchiuderli tra parentesi* (allo stesso modo non c'è bisogno di inserire il carattere "|": esso indica semplicemente che dovete scegliere uno degli argomenti che si trovano ai lati. Nel caso della figura 3-1 dovreste scegliere tra la virgola e il punto e virgola). In un enunciato PRINT entrambi gli argomenti sono facoltativi; infatti la parola chiave PRINT può essere usata anche da sola.

Esercizio: Uso di un enunciato PRINT con argomenti

1. Selezionate il comando Nuovo dal menu File

2. Scrivete il programma seguente ed eseguitelo:

```
CLS
PRINT "Questo enunciato usa un argomento"
PRINT
PRINT "Anche questo enunciato usa un argomento"
```

Il vostro schermo apparirà in questo modo al momento dell'esecuzione:

```
Questo enunciato usa un argomento

Anche questo enunciato usa un argomento
```

Come potete vedere, un enunciato PRINT non corredato da alcun argomento stampa una riga vuota (si noti che non basta premere Invio per inserirne una). Vediamo ora i tre tipi di argomenti che possono essere usati con l'enunciato PRINT: testo, espressioni numeriche e funzioni.

Uso di testo come argomento

Nell'ultimo capitolo abbiamo usato del testo tra virgolette ("Marco, sono preoccupato per Maria") come argomento *listaespressione* dell'enunciato PRINT. Questo testo viene detto anche *stringa*: una stringa può contenere numeri, lettere, spazi, segni di interpunzione — con l'eccezione delle doppie virgolette che vengono interpretate da QBasic come indicatori di fine stringa.

Uso di espressioni numeriche come argomento

Un altro possibile argomento listaespressione è un'*espressione numerica*, che può essere rappresentata da un numero, un'equazione matematica o, come vedremo nel prossimo capitolo, da uno speciale tipo di parola detta *variabile numerica*. Le espressioni numeriche non debbono essere racchiuse tra virgolette.

Esercizio: Uso di espressioni numeriche come argomenti

1. Modificate il programma appena scritto in modo che contenga soltanto il seguente enunciato PRINT:

```
CLS
PRINT 42
```


2. Eseguite il programma e avrete questo risultato sullo schermo:

```
42
```

Uso di funzioni come argomenti

Un terzo argomento listaespressione è una funzione di QBasic, che esegue generalmente un compito "dietro le quinte", restituendo informazioni che possono essere usate nel programma. Anche le funzioni non devono essere racchiuse tra virgolette.

Esercizio: Uso di funzioni come argomenti

QBasic contiene due funzioni, dette DATE$ e TIME$, che mostrano la data e l'ora corrente sulla base dell'orologio interno al vostro computer. Non è possibile usare queste funzioni da sole (in altre parole non le si può collocare su una riga e aspettarsi che succeda qualcosa) ma si possono usare come argomenti di un enunciato PRINT.

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete questo programma ed eseguitelo:

```
CLS
PRINT DATE$
PRINT TIME$
```

Lo schermo apparirà in questo modo al momento dell'esecuzione:

```
25-03-1991
21:13:09
```

È ovvio che la data e l'ora in questo esempio non sono uguali a quelle che voi vedrete in output sul vostro schermo, ma il principio di fondo rimane: le funzioni DATE$ e TIME$ hanno eseguito un certo compito e hanno restituito un certo valore. Questa è la bellezza delle funzioni di QBasic: eseguono il compito cui sono preposte senza che ci si debba preoccupare dei particolari.

Stampare più elementi con l'istruzione PRINT

Fino ad ora abbiamo usato un solo argomento (per esempio una singola stringa o una singola funzione) per ogni enunciato PRINT. È possibile tuttavia usare più argomenti in un solo enunciato PRINT, purché si usi uno speciale carattere, detto separatore, per separarli; PRINT riconosce due caratteri di separazione: virgola e punto e virgola.

La virgola

In programmazione si può considerare la virgola come l'equivalente del tasto Tab. Quando PRINT incontra una virgola stampa il valore dell'argomento successivo all'inizio della successiva zona di stampa (le zone di stampa sono l'equivalente delle tabulazioni; la loro lunghezza è di 14 caratteri). Questa capacità di stampare in collocazioni specifiche fa della virgola la scelta più comune quando si vogliono stampare informazioni in colonna.

Esercizio: uso della virgola come separatore

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma per vedere come agisce la virgola usata come separatore:

```
CLS
PRINT "Le", "virgole", "separano", "argomenti"
PRINT "Le", "virgole", "allineano", "argomenti"
```

Il vostro schermo output apparirà in questo modo:

```
Le  virgole  separano  argomenti
Le  virgole  allineano  argomenti
```

NOTA: Se non avete inserito uno spazio dopo ogni virgola, QBasic lo inserisce automaticamente quando premete Invio alla fine della riga. Uno spazio viene sempre inserito dopo un separatore per dare un'immagine più ordinata al programma.

Le virgole si possono usare anche da sole, cioè senza che vi sia alcun argomento tra di loro, come mostra l'esercizio successivo.

Esercizio: posizionamento degli argomenti con le virgole

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che usa le virgole per posizionare certi valori al centro dello schermo:

```
CLS
PRINT , , "Uso della virgola per"
PRINT , , "separare gli argomenti"
```

Il vostro schermo output apparirà in questo modo:

```
                            Uso della virgola per
                            separare gli argomenti
```

Il punto e virgola

Quando l'istruzione PRINT incontra un punto e virgola stampa l'argomento successivo immediatamente dopo quello appena stampato, senza inserire alcuno spazio tra gli argomenti; diventa perciò necessario mettere gli spazi se si vuole che questi vengano inseriti.

Esercizio: Uso del punto e virgola come separatore

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che usa il punto e virgola come separatore, con e senza l'inserimento di spazi:

```
CLS
PRINT "Il"; "punto"; "e"; "virgola"
PRINT "Il "; "punto "; "e "; "virgola "
```

Il vostro schermo output apparirà in questo modo:

```
Ilpuntoevirgola
Il punto e virgola
```

Uso della virgola e del punto e virgola insieme

Naturalmente è possibile usare i due separatori insieme su una stessa riga; virgole e punti e virgole possono essere usati contemporaneamente, continuando a svolgere le loro funzioni.

Esercizio: Uso della virgola e del punto e virgola insieme

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che usa la virgola e il punto e virgola sulla stessa riga:

```
CLS
PRINT "Queste"; "parole"; "sono"; "unite", "Queste", "sono", "separate"
```

Il vostro schermo output apparirà in questo modo:

```
Questeparolesonounite        Queste     sono     separate
```

Uso di un separatore alla fine di un enunciato PRINT

Quando si inserisce una virgola o un punto e virgola alla fine di un enunciato PRINT, si indica il punto in cui dovrà apparire il risultato dell'enunciato PRINT successivo:

- Una virgola stampa il risultato dell'enunciato PRINT successivo all'inizio della successiva area di stampa.
- Un punto e virgola stampa il risultato dell'enunciato PRINT successivo immediatamente dopo quello dell'enunciato corrente.

Esercizio: Uso di un separatore alla fine di un enunciato PRINT

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che usa il punto e virgola alla fine di un enunciato PRINT:

```
CLS
PRINT "Questa è la prima riga";
PRINT "e questa è la seconda"
```


Il risultato sarà:

```
Questa è la prima riga e questa è la seconda
```

3. Ora sostituite il punto e virgola con una virgola ed eseguite il programma di nuovo.

NOTA: Se si usa il punto e virgola alla fine di un enunciato PRINT e non ci sono altre enunciati di questo tipo, l'uso di questo separatore non produce alcun effetto.

SOMMARIO

In questo capitolo abbiamo approfondito le istruzioni di programmazione di QBasic e abbiamo visto varie opzioni di sintassi che rendono gli enunciati di QBasic più versatili. Nel prossimo capitolo vedremo due importanti elementi di QBasic in grado di rendere i programmi ancora più flessibili: variabili e operatori.

DOMANDE ED ESERCIZI

1. Descrivete brevemente la differenza tra un enunciato e una funzione.
2. Quali tra queste parole chiave sono enunciati e quali sono funzioni?

```
BEEP      DATE$     TIME$
CLS       PRINT
```

3. In una riga di sintassi qual è il significato delle parentesi quadre attorno a un oggetto? Che cosa significa il carattere "|"?
4. Che cos'è un argomento?
5. Qual è la differenza tra una stringa e un'espressione numerica?
6. Quale sarà il risultato di questi due enunciati PRINT?

```
PRINT "Ti", ; "amo"
PRINT "Ti"; , "amo"
```

7. Che cosa succede quando si inserisce un punto e virgola o una virgola alla fine di un'istruzione PRINT?

CAPITOLO 4

Variabili e operatori di QBasic

Durante la programmazione può essere necessario stampare più di una volta certi numeri o stringhe di caratteri. QBasic soddisfa quest'esigenza con un metodo molto semplice che non richiede una gran quantità di digitazioni. Con QBasic è infatti possibile conservare dati e usarli quando necessario semplicemente utilizzando il nome del luogo in cui questi dati sono conservati; a questo luogo potete attribuire un nome in base al tipo e alla dimensione dei dati che contenuti.

CHE COSA È UTILE CONSERVARE?

Le variabili sono di due tipi: *stringhe* e *numeriche*. Una variabile stringa è un nome che indica il posto in cui è conservata una stringa; una variabile numerica è un nome che indica il posto in cui è conservato un numero. Le variabili numeriche possono essere suddivise in molti sotto tipi. La figura 4-1 mostra i tipi di variabili che possono essere usati in QBasic.

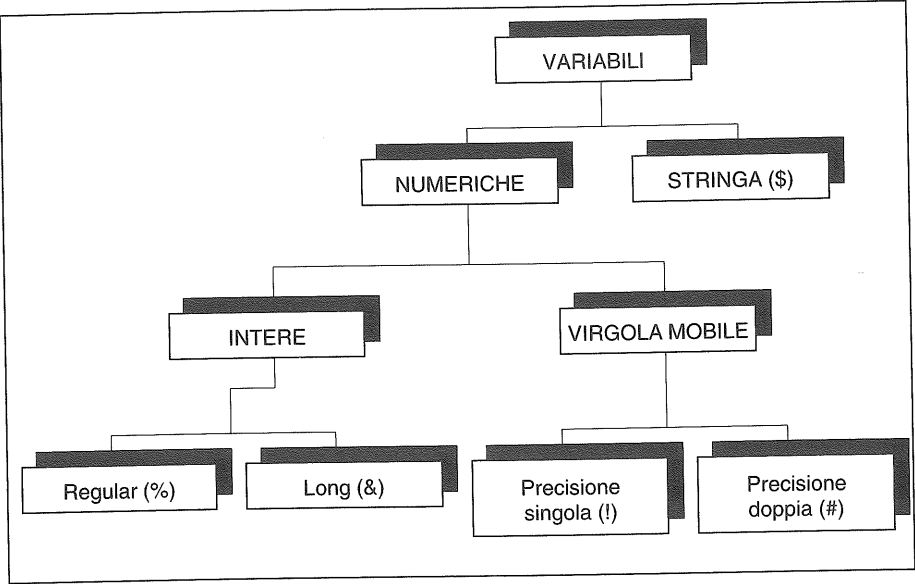



Figura 4-1. *Quadro generale dei tipi di variabili utilizzabili in QBasic.*

Nel leggere questo capitolo cercate di tenere a mente le seguenti domande, dal momento che le loro risposte vi potranno aiutare a determinare quale tipo di variabile è più indicata in relazione ai vostri scopi.

- **I dati da conservare sono testo o numeri?** Un testo viene conservato in una variabile stringa; un numero in una variabile numerica.
- **Se il dato da conservare è un numero, contiene un separatore decimale?** Un intero — cioè un numero senza separatore decimale — viene conservato in una variabile intera. Un numero con un separatore decimale viene conservato in una variabile in virgola mobile.
- **Quanto è grande il numero da conservare?** La dimensione gioca un ruolo importante nel determinare quale variabile usare.

DESCRIZIONE GENERALE SULL'USO DELLE VARIABILI

Quando si decide di usare una variabile bisogna dichiararlo nel programma, il che significa che bisogna dare a QBasic tre informazioni:

- Il nome della variabile
- Il tipo della variabile
- Il valore della variabile

La figura 4-2 mostra gli elementi di una dichiarazione, attraverso l'esempio della dichiarazione di una variabile stringa.

La sezione che segue descrive come usare ciascuno di questi elementi.

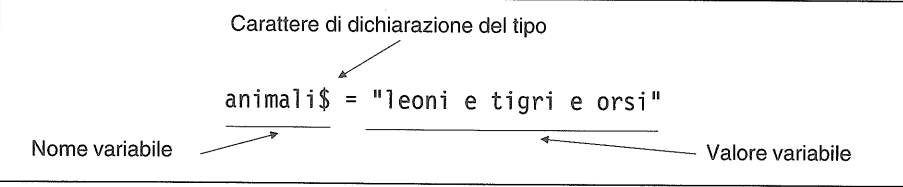



Figura 4-2. *Gli elementi della dichiarazione di una variabile.*

Dare il nome a una variabile

Le regole e i suggerimenti che seguono vi aiuteranno nello scegliere i nomi più appropriati da attribuire alle variabili di QBasic:

- Il nome di una variabile può essere costituito da un massimo di 40 caratteri.

- Il nome di una variabile può essere costituito da una qualsiasi combinazione di lettere maiuscole e minuscole (tenete presente, però, che nomi tutti in maiuscole potrebbero confondersi con le parole chiave di QBasic).
- Il nome di una variabile non può coincidere con una parola chiave di QBasic, come PRINT o BEEP.
- Il carattere per indicare la dichiarazione del tipo di variabile dev'essere l'ultimo carattere del nome della variabile (si veda "Dichiarazione del tipo di variabile").
- Più i nomi delle variabili sono descrittivi meglio è; per esempio questi nomi di variabili stringa, Nome$ e Cognome$, lasciano pochi dubbi sul tipo di informazione in esse contenute.

NOTA: QBasic è molto attento all'uso che viene fatto delle lettere maiuscole e minuscole; così sensibile che se si digita il nome di una variabile con una diversa combinazione di maiuscole e minuscole esso modifica i nomi precedenti, usando la nuova combinazione.

Dichiarazione del tipo di variabile

Nella figura 4-2 si può notare che il carattere finale del nome della variabile coincide con il carattere di dichiarazione del tipo, che specifica a QBasic che tipo di variabile si sta dichiarando. Ogni tipo di variabile ha un proprio simbolo:

Se la variabile è questa	Usa questo carattere di dichiarazione del tipo
Stringa	$
Intero	
Regolare	%
Lungo	&
Virgola mobile	
Precisione singola	!
Precisione doppia	#

Quando QBasic incontra il carattere che specifica il tipo nel nome della variabile, sa esattamente che genere di dati vi sono conservati.

Dichiarazione del valore della variabile

Quando si dichiara il valore di una variabile bisogna assicurarsi che il valore concordi con il tipo di variabile specificato. Per esempio, una variabile intera non potrebbe contenere un valore stringa.

E adesso?

Dopo aver visto la struttura generale della dichiarazione di una variabile possiamo cominciare a vedere i diversi tipi di variabili esistenti. Cominceremo con le variabili stringa e numeriche per poi vedere come ottenere informazioni dalla tastiera e come conservarle nelle variabili. La parte finale di questo capitolo descrive il funzionamento di QBasic in relazione a calcoli matematici.

VARIABILI STRINGA IN QBASIC

Il nome di una variabile stringa rappresenta il luogo in cui viene immagazzinata una stringa di testo che si vuole usare nuovamente all'interno del programma.

Per dichiarare una variabile stringa bisogna inserire il carattere dollaro ($) alla fine del nome della variabile. La stringa va racchiusa tra virgolette nello stesso modo visto per un enunciato PRINT; per esempio:

```
AutoriUsa$ = "Kerouac, Ginsberg, Ferlinghetti"
```

La dimensione e il contenuto delle variabili stringa possono cambiare all'interno del programma, come vedremo nel capitolo 9.

Esercizio: Dichiarare e usare una variabile stringa

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che assegna un valore a una variabile stringa e poi la stampa:

```
CLS
politici$ = "Moro, Berlinguer, Nenni"
PRINT politici$
```

L'output del programma sarà questo:

```
Moro, Berlinguer, Nenni
```

VARIABILI NUMERICHE IN QBASIC

QBasic usa due tipi di variabili numeriche: le variabili numeriche intere e le variabili numerica, in virgola mobile. Ciascun tipo prevede dei sotto tipi per numeri di dimensioni diverse (ricordate che la dimensione del numero può aiutarvi a determinare che tipo di variabile usare). La figura 4-3 mostra le relazioni esistenti tra i diversi tipi di variabili numeriche in QBasic.

Per prima cosa esamineremo i due tipi di variabili intere e quando usarle; poi passeremo all'analisi dei due tipi di variabili in virgola mobile.

Variabili intere

QBasic usa due tipi di variabili intere: l'intero di tipo *regular* e l'intero di tipo *long*. La differenza tra i due consiste nella dimensione del numero intero che ciascuno di essi può rappresentare.

QBasic e i numeri

Ciascuno di noi passa una buona parte del suo tempo lavorando con numeri (si pensi al controllo delle spese di casa o alla preparazione di un bilancio). Nessuno fa particolare attenzione al fatto che vi sono diversi tipi di numeri e, per molti di noi, i nomi "propri" dei numeri — interi, decimali etc. — non sono altro che un ricordo della matematica imparata a scuola. Per QBasic invece le differenze tra due particolari tipi di numeri sono cruciali:

- Un intero è un numero senza decimali.
- Un numero in virgola mobile è un numero che contiene un separatore decimale.

Variabili intere regolari

Una variabile intera di tipo *regular* può contenere qualsiasi numero intero compreso tra -32.768 e 32.767. Per dichiarare una variabile intera di tipo *regular* si usa il simbolo di percentuale (%) come carattere di dichiarazione del tipo da usare alla fine del nome della variabile. Per esempio:

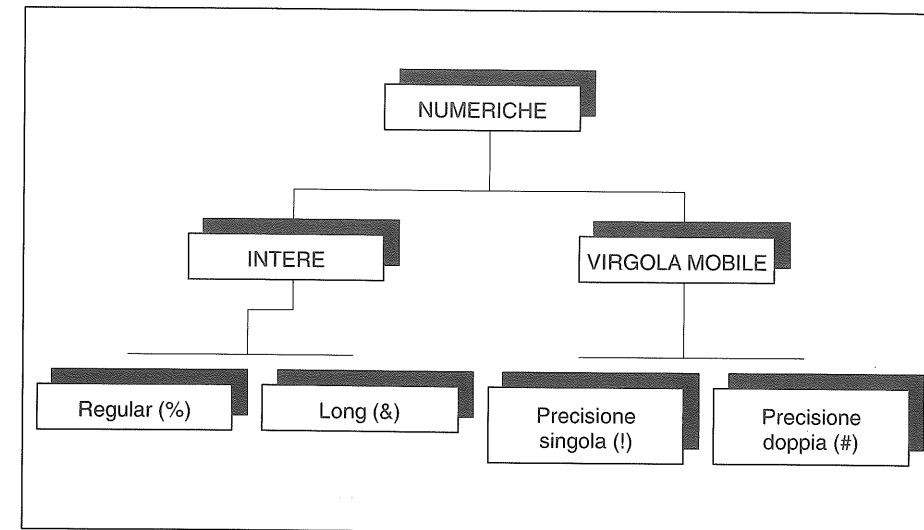
```
enricoMogli% = 6
```


Figura 4-3. *Variabili numeriche in QBasic.*

Esercizio: Dichiarare e usare una variabile intera di tipo regular

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che dichiara e stampa una variabile intera di tipo *regular*:

```
CLS
giorniNatale% = 12
PRINT giorniNatale%
```

La schermata di output apparirà in questo modo:

```
 12
```

Uno spazio addizionale per i numeri

Se usate l'enunciato PRINT per visualizzare dei numeri potrete notare che questi vengono fatti precedere da uno spazio; lo spazio serve a QBasic per inserire un segno meno quando il numero è negativo. Perciò se il numero è positivo QBasic visualizza uno spazio vuoto.

Variabili intere di tipo *long*

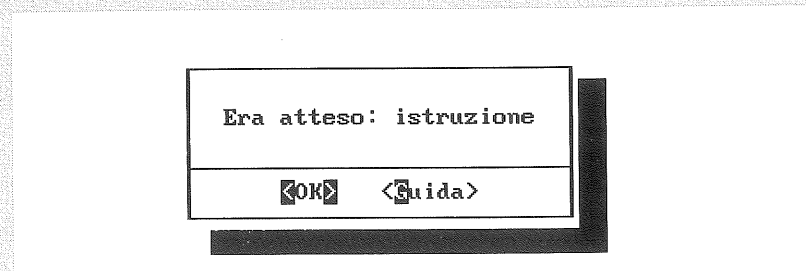
Se si deve usare un numero intero non compreso nell'intervallo indicato precedentemente bisogna utilizzare una variabile intera di tipo *long*, la quale può rappresentare un numero intero compreso tra –2.147.483.648 e 2.147.483.647.

Per dichiarare una variabile intera di tipo *long* si usa il simbolo di E commerciale (&) come carattere di dichiarazione del tipo da inserire alla fine della variabile. Per esempio:

```
Popolazione& = 175000
```

Non si possono usare le virgole nei numeri

Sebbene questo sia tipico dell'impostazione numerica americana è bene ricordare che siccome la virgola viene usata da QBasic come separatore di caratteri, non può essere impiegata all'interno di espressioni numeriche e si dovrà usare il punto come separatore decimale.

Se si inserisce una virgola in un'espressione numerica all'interno del programma, QBasic mostra un messaggio di errore:

Allo stesso modo, se si digita un numero contenente virgole in fase di esecuzione del programma, QBasic chiede di digitarlo di nuovo.

Esercizio: Dichiarazione e uso di una variabile intera di tipo long

1. Selezionate il comando Nuovo dal menu File.

2. Scrivete ed eseguite questo programma che dichiara e stampa una variabile intera di tipo *long*:


```
CLS
cadutalibera& = 84700
PRINT "La più alta caduta libera, in piedi, fu"; cadutalibera&
```

Il risultato apparirà così:

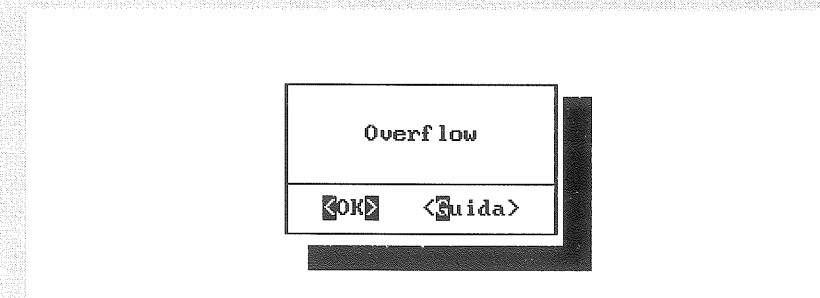
```
La più alta caduta libera, in piedi, fu 84700
```

Numeri in virgola mobile

Fino ad ora abbiamo visto solo numeri interi. Molto spesso, però, si devono usare numeri che hanno una parte frazionale: cioè un numero seguito da un separatore decimale e l'espressione decimale della frazione.

QBasic fornisce due tipi di variabili per rappresentare numeri in virgola mobile: virgola mobile a precisione singola e a precisione doppia. La differenza principale fra i due tipi di variabili consiste nell'accuratezza con cui possono rappresentare un certo numero contenente una virgola mobile.

Uso di una variabile di dimensione sbagliata

Come abbiamo visto, QBasic richiede che si faccia molta attenzione alla dimensione di un numero. Se si attribuisce a una variabile un numero al di fuori dell'intervallo che è in grado di rappresentare, QBasic mostra una finestra di dialogo per indicare che il numero non è ammesso per quel tipo di variabile:

Per evitare questo tipo di errore si deve fare molta attenzione nella scelta delle variabili e mostrare all'utente del programma l'intervallo di valori consentiti.

- Una variabile in virgola mobile a precisione singola può rappresentare un numero composto da un massimo di 7 cifre e il punto decimale si può trovare in una qualsiasi posizione tra le cifre.
- Una variabile in virgola mobile a precisione doppia può rappresentare un numero composto da un massimo di 15 cifre, con il punto decimale collocabile, anche in questo caso, in una posizione qualsiasi.

La tabella seguente mostra alcuni esempi di valori in virgola mobile:

Precisione singola	Precisione doppia
412.002	1.000032478
19246.34	4280000.0055
.00025	7160.0000000005
657926.3	.10000000001

Variabili in virgola mobile a precisione singola

Per dichiarare una variabile in virgola mobile a precisione singola bisogna usare il punto esclamativo (!) come carattere di dichiarazione del tipo alla fine del nome della variabile. Per esempio:

```
lpveloc! = 33.3333
```

Esercizio: Dichiarazione e uso di una variabile in virgola mobile a precisione singola

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che dichiara una variabile virgola mobile a precisione singola e ne mostra il valore.

```
CLS
tvPrezzo! = 2499999
PRINT "Questo televisore costa "; tvPrezzo!
```

L'output del programma sarà:

```
Questo televisore costa  2499999
```


Variabili in virgola mobile a precisione doppia

Le variabili in virgola mobile a precisione doppia (che possono rappresentare un numero composto al massimo di 15 cifre) sono particolarmente utili per scopi scientifici che richiedono numeri molto precisi.

Per dichiarare una variabile in virgola mobile a precisione doppia bisogna usare il simbolo # come carattere di dichiarazione del tipo alla fine del nome della variabile. Per esempio:

```
piGreco! = 3.141592653589793
```

Esercizio: Dichiarazione e uso di una variabile in virgola mobile a precisione doppia

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma che dichiara e stampa una variabile in virgola mobile a precisione doppia e ne mostra il valore:

```
CLS
piGreco! = 3.141592653589793
PRINT "Il valore di pi greco è "; piGreco!
```

L'output del programma apparirà in questo modo:

```
Il valore di pi greco è 3.141592653589793
```

Come scegliere la variabile numerica da usare?

Abbiamo visto che QBasic mette a disposizione quattro tipi di variabili numeriche: intera *regular* e *long*, in virgola mobile a precisione singola e virgola mobile a precisione doppia. Quando si vuole assegnare un valore numerico al nome di una variabile del programma bisognerà scegliere tra queste il tipo di variabile più appropriato; per far ciò può essere utile porsi le seguenti domande:

- Il numero è intero e resterà tale nel corso dell'intero programma? Se la risposta è positiva si userà una variabile intera.
- Il numero contiene una virgola decimale? O, meglio ancora, il numero potrebbe contenere una frazione nel resto del programma? Se la risposta è positiva si userà una variabile in virgola mobile.

Una volta determinato il tipo di variabile — intera o in virgola mobile — bisogna sceglierne la dimensione.

Scegliere la dimensione appropriata per una variabile

A prima vista si potrebbe pensare che le variabili intere di tipo *long* e in virgola mobile a doppia precisione siano la scelta migliore in ogni caso, perché possono contenere numeri più grandi, riducendo così il rischio di ottenere un messaggio di "Overflow" (che indica che il numero è al di fuori dell'intervallo ammesso dalla variabile). Purtroppo anche queste variabili più grandi presentano degli svantaggi che bisogna conoscere: se da un lato offrono maggior spazio per la conservazione di valori numerici, dall'altro richiedono una maggior quantità di memoria. Usare queste variabili quando non è necessario è un po' come conservare un sacchetto di grano in un silos! La memoria di un computer è come un pezzo di terra: se si costruisce un silos ogni volta che si vuole immagazzinare un sacco di grano, alla fine si resta senza terra sul quale coltivarlo. Vi consigliamo perciò di usare attentamente le risorse disponibili, scegliendo in modo oculato il tipo di variabile da usare.

Cambiamento dei contenuti di una variabile

È facile capire l'utilità delle variabili: i loro contenuti possono variare in base alle esigenze del programma o dell'utente. È infatti possibile cambiarne i contenuti semplicemente assegnando loro un nuovo valore. Ricordate comunque che il valore corrente di una variabile è quello che le è stato assegnato per ultimo.

Esercizio: Cambiamento del valore di una variabile

1. Selezionate il comando Nuovo dal menu File.

2. Scrivete ed eseguite questo programma che dichiara e poi modifica il valore di una variabile:

```
CLS
frutta$ = "mela"
PRINT frutta$
frutta$ = "arancia"
PRINT frutta$
frutta$ = "pera"
```



```
PRINT frutta$
```

La schermata di output apparirà in questo modo:

```
mela
arancia
pera
```

USO DI INFORMAZIONI FORNITE DALL'UTENTE IN UNA VARIABILE

Dopo aver visto le variabili, cercheremo ora di analizzare un'area della programmazione in QBasic che si basa molto sull'uso di variabili e, in particolare, sull'ottenimento di caratteri da parte dell'utente del programma. Questo processo prende talvolta il nome di "lettura di caratteri dalla tastiera".

Lettura di caratteri con l'enunciato INPUT

L'enunciato di QBasic più comunemente usato per la lettura di caratteri dalla tastiera è INPUT; quest'enunciato non può essere usato da solo ma dev'essere seguito dal nome della variabile (incluso il tipo) che si vuole assegnare ai caratteri digitati dall'utente.

Esercizio: Uso dell'enunciato INPUT

1. Selezionate il comando Nuovo dal menu File.

2. Scrivete ed eseguite questo programma:

```
CLS
PRINT "Digitare una parola e premere Invio"
INPUT parola$
PRINT "La parola digitata è "; parola$
```

Ecco il risultato:

```
Digitare una parola e premere Invio
? Giraffa
La parola digitata è Giraffa
```

Diamo un'occhiata a ciò che è accaduto:

- Per prima cosa il programma ha pulito lo schermo e ha usato l'enunciato PRINT per dire all'utente di digitare una parola.
- Poi, seguendo le istruzioni dell'enunciato PRINT, abbiamo digitato una parola e premuto Invio. Questa è stata assegnata alla variabile parola$ dall'enunciato INPUT.
- L'ultimo enunciato PRINT usa sia la stringa sia la variabile parola$ per visualizzare ciò che era stato digitato.

Richiesta di input all'utente

Il programma appena descritto usa l'enunciato PRINT per chiedere all'utente di fornire alcune informazioni, ma è possibile inserire la richiesta direttamente nell'enunciato INPUT. In questo modo INPUT viene a svolgere un duplice compito: richiede l'input all'utente e legge ciò che è stato digitato.

Esercizio: Collocare un messaggio all'interno di INPUT

Cambiate il programma precedente in questo modo:

```
CLS
INPUT "Digitare una parola e premere Invio"; parola$
PRINT "La parola digitata è "; parola$
```

Dopo aver eseguito il programma la schermata di output dovrebbe apparire in questo modo:

```
Digitare una parola e premere Invio? Giraffa
La parola digitata è Giraffa
```

Notate le differenze tra questo programma e quello precedente:

- Siccome la richiesta fa parte dell'enunciato INPUT, il cursore rimane sulla stessa riga (e non in quella sottostante) in attesa di una risposta.
- L'enunciato INPUT mostra un punto di domanda alla fine del messaggio (per eliminarlo basta sostituire con una virgola il punto e virgola che si trova alla fine del messaggio INPUT ed eseguire nuovamente il programma).

Esercizio: Aggiungere un tocco di creatività all'enunciato INPUT

Ora che abbiamo visto i fondamenti delle variabili e come ottenere informazioni da tastiera con l'enunciato INPUT, cerchiamo di aggiungere un po' di creatività al programma:

1. Selezionate il comando Nuovo dal menu File.
2. Scrivete ed eseguite questo programma:

```
CLS
INPUT "Digita il tuo nome e premi Invio: ", nome$
PRINT
INPUT "Quanti anni hai (prometto di non dirlo a nessuno)? ", anni%
PRINT
INPUT "Quante lire hai in tasca?", soldi!
PRINT
PRINT "Grazie, "; nome$; ". Hai detto che la tua età è"; anni%
PRINT "anni e che hai"; soldi!; "lire in tasca."
```

Dopo aver eseguito il programma, la schermata di output dovrebbe apparire in questo modo:

```
Digita il tuo nome e premi Invio: Lorenzo

Quanti anni hai (prometto di non dirlo a nessuno)? 30

Quante lire hai in tasca? 10000

Grazie, Lorenzo. Hai detto che la tua età è 30
anni e che hai 10000 lire in tasca.
```

OPERATORI MATEMATICI DI QBASIC

QBasic mette a disposizione diversi simboli che si possono usare per eseguire calcoli matematici all'interno dei programmi. Questi simboli, detti operatori, permettono di eseguire addizioni, sottrazioni, moltiplicazioni e divisioni.

Molti operatori QBasic sono simili a quelli normalmente usati per i calcoli matematici; per esempio si usa il segno più (+) per l'addizione e quello meno (–) per la sottrazione. Alcuni operatori sono invece rappresentati da simboli speciali. La tabella che segue elenca tutti gli operatori che si possono usare in QBasic:

Operatore	Operazione matematica
+	Addizione
–	Sottrazione
*	Moltiplicazione
/	Divisione
\	Intero della divisione
MOD	Resto della divisione
^	Esponente (elevazione a potenza)

Gli ultimi tre operatori (\ , MOD e ^) sono operatori a scopo specifico, come vedremo tra poco.

Lavorare con gli operatori di QBasic

L'uso degli operatori per eseguire calcoli matematici è molto semplice: per esempio, per sommare tra loro i numeri 12 e 16 basta scrivere

```
12+16
```

Tuttavia non si può semplicemente collocare un'operazione matematica come questa su una riga da sola. Ci sono due cose che si possono fare:

- Assegnare il risultato dell'operazione matematica a una variabile numerica:

```
totale% = 12+16      'Assegna il risultato a una variabile
```

- Usare il risultato dell'operazione matematica come argomento per un enunciato di QBasic:

```
PRINT 12+16          'Stampa il risultato direttamente
```


Aggiungere commenti ai programmi

QBasic permette di inserire dei commenti all'interno del programma, a scopo di informazione sulle varie azioni di volta in volta eseguite. I commenti vanno fatti precedere dall'enunciato REM o dal simbolo " ' " e servono a uso esclusivo del programmatore; essi infatti non appaiono quando il programma viene eseguito. Provate a scrivere ed eseguire questo programma per vedere come funziona l'enunciato REM:

```
REM ESEMPIO.BAS
REM Programma dimostrativo sull'uso dei commenti.
REM
REM Programmatori: Matteo e Luca
REM Data: 3 marzo 1991

CLS

PRINT "Questo è un programma con commenti"
```

Il programma produrrà il seguente risultato:

```
Questo è un programma con commenti
```

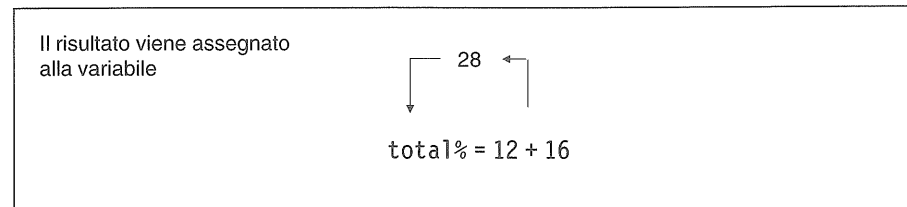
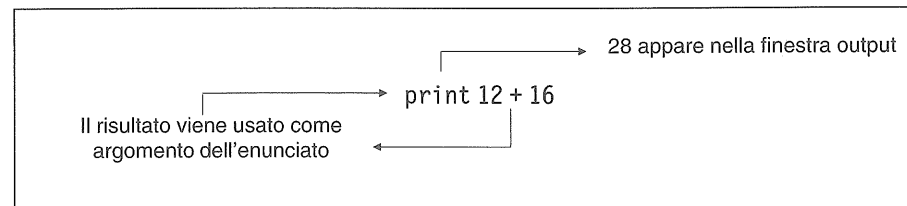
Il simbolo " ' " può essere usato come simbolo meno appariscente ma che svolge la stessa funzione dell'enunciato REM:

```
' ESEMPIO.BAS
' Programma dimostrativo sull'uso dei commenti.
'
' Programmatori: Matteo e Luca
' Data: 3 marzo 1991

CLS

PRINT "Questo è un programma con commenti"
```

Il programma produrrà lo stesso risultato:

```
Questo è un programma con commenti
```

In questo libro useremo questo secondo metodo per i commenti. È consigliabile includere sempre dei commenti sia per proprio uso, sia per altri utenti che cercassero di capire la struttura dei vostri programmi.

Si noti che quando si assegna un'operazione matematica a una variabile, ciò che in realtà si assegna è il risultato dell'operazione, come si vede qui sotto:

Allo stesso modo quando si assegna un'operazione matematica all'argomento di un enunciato ciò che in realtà si usa è il risultato dell'operazione:

Esercizio: Lavorare con gli operatori di QBasic

Nel capitolo 2 abbiamo visto che la Finestra Immediato è molto utile per provare le righe di programma prima di inserirle; negli esercizi sull'uso degli operatori di QBasic useremo questa finestra.

1. Premete F6 per attivare la Finestra Immediato e poi inseritevi questa riga di programma:

```
CLS
```

2. Premete un tasto qualsiasi per attivare la Finestra Immediato e inseritevi questa riga:

```
PRINT 7*7
```

La schermata di output mostrerà il risultato dell'enunciato PRINT:

```
49
```


Uso della Finestra Immediato

La Finestra Immediato vi permette di controllare righe di programma individuali prima di inserirle nella Finestra di digitazione. Queste osservazioni possono aiutarvi a usare efficientemente questa finestra:

- È possibile eseguire una sola riga di programma alla volta.
- Per eseguire una riga nella Finestra Immediato basta premere Invio e non c'è bisogno di scegliere Avvia.
- Le righe eseguite nella Finestra Immediato non possono essere salvate a meno che non vengano spostate all'interno della Finestra di digitazione.

3. Riattivate la Finestra Immediato e inseritevi questa riga di programma:

```
PRINT 75/6
```

Anche questa volta l'output visualizzerà il risultato dell'enunciato PRINT:

```
12.5
```

Gli operatori \ , MOD e ^

QBasic ha tre operatori con scopo specifico: \ , MOD e ^ : addizione, sottrazione, moltiplicazione e divisione.

L'operatore \ (intero della divisione)

L'operatore intero della divisione (\) funziona esattamente come un operatore di divisione normale (/) con la differenza che quando un numero viene diviso per un altro la parte frazionale del risultato viene scartata, mantenendo solo la parte intera.

Esercizio: Uso dell'operatore (\)

Digitate la riga seguente nella Finestra Immediato:

```
PRINT 5\2
```

Lo schermo visualizzerà il risultato dell'operazione:

```
2
```

Il risultato normale della divisione sarebbe 2.5 o 2 col resto di 1, ma l'intero della divisione scarta il resto dando come risultato solo l'intero, cioè 2.

L'operatore MOD

MOD è a tutti gli effetti un operatore di QBasic nonostante le apparenze. L'operatore MOD viene anche chiamato operatore resto della divisione perché il suo risultato è l'opposto dell'operatore (\): ciò significa che quando si divide un numero per un altro usando questo operatore, QBasic scarta la parte intera del risultato restituendo solamente il resto.

Esercizio: Uso dell'operatore MOD

Digitate la riga seguente nella Finestra Immediato:

```
PRINT 5 MOD 2
```

Il risultato dell'operazione che appare nello schermo output sarà il seguente:

```
1
```

Anche in questo caso il risultato dell'operazione sarebbe 2 con il resto di 1, ma siccome viene usato l'operatore MOD, QBasic scarta la parte intera del risultato conservando solo il resto.

L'operatore ^ (esponente)

L'operatore esponente (^) vi consente di eseguire operazioni di elevamento a potenza; per esempio, l'equivalente di 10^3 (dieci alla terza) viene espresso in questo modo:

```
10 ^ 3
```


Regole diverse per la divisione

L'operatore della divisione normale (/) restituisce il risultato completo; per esempio l'enunciato seguente restituirebbe lo stesso risultato di una normale calcolatrice:

```
PRINT 5/2
```

Gli operatori \ e MOD usano il metodo di divisione più tradizionale, distinguendo tra parte intera e resto.

Esercizio: Uso dell'operatore ^

Digitate la riga seguente nella Finestra Immediato:

```
PRINT 10 ^ 3
```

Il risultato sarà il seguente:

```
1000
```

Espressioni numeriche

Un'espressione numerica, comunemente detta formula, è una combinazione di numeri, variabili, operatori e funzioni numeriche che insieme producono un risultato. Gli esercizi sulla moltiplicazione, sulla divisione e sull'elevamento a potenza possono essere considerati come semplici esempi di espressioni numeriche.

QBasic permette di creare un'ampia gamma di espressioni numeriche, da quelle più semplici a quelle più elaborate; la creazione di queste espressioni non è complicata e basta tenere a mente alcune semplici regole per essere in grado di effettuare qualsiasi tipo di calcolo.

Usare più di un operatore

Negli esempi appena visti si usavano operatori singoli ma QBasic permette anche di usare più di un operatore nella stessa espressione numerica per effettuare più calcoli allo stesso tempo.

Esercizio: Uso di più operatori

Digitate la riga seguente nella Finestra Immediato:

```
PRINT 14 + 26 + 15.75 - 33.2
```

Il risultato dell'operazione che appare nello schermo di output sarà quello mostrato di seguito:

```
22.55
```

La Finestra Immediato può quindi essere molto utile anche come calcolatrice!

Ordine di calcolo

QBasic rispetta un rigoroso insieme di regole nel calcolare un'espressione numerica che contiene più di un operatore. Si consideri il seguente esempio:

```
3 + 4 * 5
```

Qual è il risultato? In effetti ci sono due diversi risultati a seconda di come il calcolo viene eseguito: se si esegue prima l'addizione il risultato è 35, altrimenti (cioè se si esegue prima la moltiplicazione) è 23.

Per evitare confusione, QBasic esegue i calcoli nell'ordine seguente:

- L'elevamento a potenza (^) viene eseguita per prima
- La moltiplicazione e la divisione (*, /,\ , MOD) per seconde.
- L'addizione e la sottrazione (+ e -) per ultime.

Queste regole non coprono però tutte le circostanze: che cosa succede per quelle operazioni i cui operatori hanno lo stesso "livello di importanza"? Per esempio:

```
3 * 5 MOD 2
```

oppure

```
100 / 4 * 3
```

In questi casi QBasic esegue i calcoli da sinistra verso destra.

Esercizio: Lavorare con l'ordine di precedenza degli operatori

Digitate la riga seguente nella Finestra Immediato:

```
PRINT 3 + 4 * 5
```

Il risultato dell'operazione sarà il seguente:

```
23
```

Poiché QBasic esegue la moltiplicazione prima dell'addizione il risultato sarà 23. Premete un tasto qualsiasi per tornare nella Finestra Immediato e scrivete questo enunciato:

```
PRINT 100 / 4 * 3
```

Il risultato dell'operazione che appare nello schermo output sarà il seguente:

```
75
```

Siccome la divisone (/) e la moltiplicazione (*) hanno lo stesso "peso" — cioè la stessa priorità — QBasic calcola l'espressione da sinistra a destra.

Uso di parentesi per controllare l'ordine di calcolo

Consideriamo l'espressione numerica precedente:

```
3 + 4 * 5
```

Come abbiamo visto, QBasic esegue la moltiplicazione prima dell'addizione, dando 23 come risultato. Che cosa succede se si vuole calcolare l'addizione prima della moltiplicazione?

QBasic permette di controllare l'ordine di calcolo di un'espressione numerica attraverso l'uso di parentesi: le operazioni tra parentesi verranno infatti calcolate prima delle altre. Nell'esempio seguente il risultato è pari a 35 perché QBasic calcola ciò che è contenuto nelle parentesi prima degli altri elementi dell'espressione:

```
(3 + 4) * 5
```

Per l'esecuzione di calcoli all'interno delle parentesi QBasic segue le stesse regole viste in precedenza. Per esempio, l'espressione seguente avrà un risultato pari a 23:

```
(3 + 4 * 5)
```

Esercizio: Uso delle parentesi per controllare l'ordine di calcolo

1. Scrivete quest'enunciato nella Finestra Immediato:

```
PRINT 3 + 4 * 5
```

Il risultato dell'output del programma sarà:

```
23
```

2. Premete un tasto qualsiasi per tornare nella Finestra Immediato e scrivete quest'altro enunciato

```
PRINT (3 + 4) * 5
```

Il risultato questa volta sarà diverso:

```
35
```

Uso di parentesi dentro altre parentesi

Per l'esecuzione di operazioni più complesse QBasic permette l'uso di parentesi annidate — cioè di parentesi inserite dentro altre parentesi.

Considerate la seguente espressione numerica:

```
2 + 3 * 4 ^ 2
```

QBasic calcolerebbe prima l'esponente, poi la moltiplicazione e, per ultima, l'addizione. Come fare nel caso si voglia eseguire prima l'addizione, poi la moltiplicazione e, infine l'esponente? Si consideri questa espressione:

```
(2 + 3 * 4) ^ 2
```

In questo caso QBasic calcolerebbe per prima cosa il contenuto delle parentesi e per ultimo l'esponente; la moltiplicazione verrebbe ancora eseguita prima dell'addizione. Per risolvere questo problema si possono usare due parentesi annidate:

```
((2 +3) * 4) ^ 2
```

QBasic calcola prima i contenuti delle parentesi interne e poi quelli delle parentesi esterne. In questo esempio l'addizione verrebbe calcolata per prima, seguita dalla moltiplicazione e dall'esponente.

Funzioni matematiche di QBasic

Come abbiamo visto nel terzo capitolo, una funzione restituisce un certo valore al programma. QBasic dispone di molte funzioni matematiche (che, come tutte le funzioni di QBasic, devono essere usate in un enunciato).

Il simbolo *n* nella tabella successiva indica il numero, la variabile numerica o l'espressione che si vuole venga eseguita dalla funzione. Si noti che la *n* è posta tra parentesi: queste devono essere usate quando l'argomento di una funzione è un valore.

Esercizio: Lavorare con funzioni numeriche

Scrivete quest'enunciato nella Finestra Immediato:

```
PRINT SQR(36)
```

Il risultato dovrebbe essere la radice quadrata di 36:

```
6
```

Funzione	Scopo
ABS(n)	Restituisce il valore assoluto di n
ATN(n)	Restituisce l'arcotangente di n in radianti
COS(n)	Restituisce il coseno dell'angolo n espresso in radianti
EXP(n)	Restituisce il logaritmo di n
SGN(n)	Restituisce –1 se n è inferiore a zero, 0 se n è zero e +1 se n è maggiore di zero
SIN(n)	Restituisce il seno dell'angolo n espresso in radianti
SQR(n)	Restituisce la radice quadrata di n
TAN(n)	Restituisce la tangente dell'angolo n espressa in radianti

Uso di funzioni matematiche in espressioni numeriche

È possibile usare funzioni matematiche in espressioni numeriche insieme con altri operatori di QBasic. Nell'esempio che segue la funzione SQR risolve un problema che riguarda un triangolo retto.

Esercizio: Uso della funzione SQR

Supponiamo di avere un palo alto 2,25 metri e un supporto di metallo piantato nel terreno a 4 metri dalla base del palo stesso (figura 4-4); vogliamo tirare un cavo dalla cima del palo al supporto metallico, ma non sappiamo esattamente quanto cavo comprare.

Si può scrivere un semplice programma in QBasic per calcolare la lunghezza usando la funzione SQR; la formula da usare per questo calcolo è la seguente:

$$\sqrt{altezza^2 + lunghezza^2}$$

1. Premete il tasto funzione F6 per attivare la Finestra di digitazione

2. Selezionate Nuovo dal menu File e scrivete questo programma:

```
CLS

PRINT "Questo programma calcola quanto cavo è necessario"
PRINT "dalla cima del palo al supporto metallico sul terreno"
PRINT
INPUT "Inserire l'altezza del palo in metri: ", altezza!
PRINT
PRINT "Inserire la lunghezza dalla base del palo"
INPUT "al supporto metallico in metri: ", lunghezza!

cavo! = SQR((altezza! ^ 2) + (lunghezza! ^ 2))

PRINT
PRINT "Avete bisogno di"; cavo!; "metri di cavo."
```

3. Eseguite il programma e inserite l'altezza e la lunghezza indicate nella figura 4-4 (2.25 e 4); il risultato sarà il seguente:

```
Questo programma calcola quanto cavo è necessario
dalla cima del palo al supporto metallico sul terreno

Inserire l'altezza del palo in metri: 2.25

Inserire la lunghezza dalla base del palo
al supporto metallico in metri: 4

Avete bisogno di 4.58939 metri di cavo.
```

Si noti che il programma usa delle variabili in virgola mobile a precisione singola; usando questo tipo di variabile non si costringe l'utente a usare soltanto numeri interi.

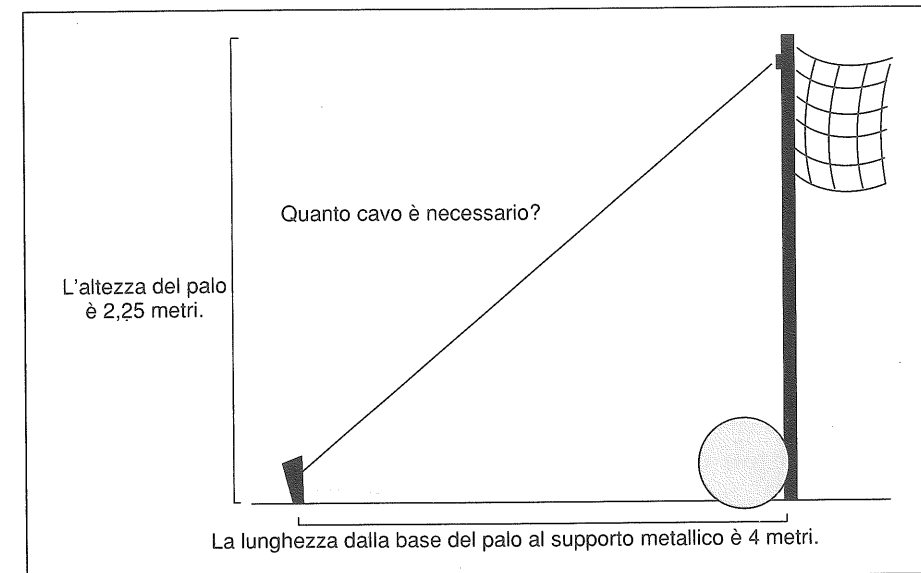



Figura 4-4. *Come trovare la lunghezza di un cavo.*

Si noti anche che la riga seguente

```
cavo! = SQR((altezza! ^ 2) + (lunghezza! ^ 2))
```

usa parentesi annidate. Viste le regole di calcolo di QBasic, si può osservare che in questo caso le parentesi non sono assolutamente necessarie e che si otterrebbe lo stesso risultato anche senza. L'uso delle parentesi può contribuire a rendere le formule più facili da leggere.

4. Eseguite il programma di nuovo, usando altri valori per l'altezza e la lunghezza (ricordate che il programma non tiene conto della lunghezza necessaria per legare il cavo al palo e al supporto metallico).

NOTA: Se avevate incluso le misure all'interno del programma, come in questo caso,

```
altezza! = 2.25
lunghezza! = 4
```

dovrete naturalmente modificare il programma stesso se volete usa-

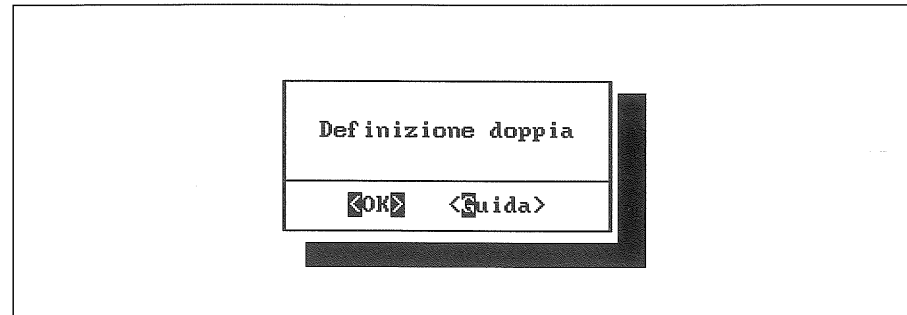
re delle misure diverse. È chiaro perciò che se si lascia che sia l'utente a inserire le misure, il programma può essere eseguito più volte senza doverlo modificare.

L'ENUNCIATO CONST

Quando si ha un valore che non cambia mai nel programma (per esempio una tassa o un tasso di sconto) lo si può dichiarare come costante con l'enunciato CONST.

L'uso di questo enunciato è molto semplice perché basta inserire la parola chiave CONST davanti alla dichiarazione della costante. Per esempio, l'enunciato seguente dice a QBasic che la costante in virgola mobile a precisione singola chiamata TASSO! rappresenterà sempre un valore pari a 0.081:

```
CONST TASSO! = 0.081
```

Se si prova ad assegnare a TASSO! un altro valore in un altro punto del programma, QBasic mostra la seguente finestra di dialogo al momento dell'esecuzione:

Esercizio: Dichiarazione di costanti con l'enunciato CONST

Supponiamo di voler calcolare il prezzo reale di uno o più oggetti — cioè il prezzo senza tasse di vendita — usando un semplice programma scritto con QBasic.

1. Selezionate il comando Nuovo dal menu File.

2. Scrivete ed eseguite il seguente programma:


```
CLS

CONS TASSO! = 0.081        ' Tassa sulle vendite

INPUT "Inserire il prezzo dell'oggetto: L", prezzo!

tassa! = prezzo! * TASSO!   ' Calcola l'ammontare della tassa
totale! = prezzo! + tassa!  ' Calcola il costo totale

PRINT
PRINT "Il costo dell'oggetto è L"; prezzo!
PRINT "Le tasse di vendita sono L"; tassa!
PRINT "-----------------------------------------"
PRINT "Il costo totale è L."; totale!
```

Il risultato che appare sullo schermo di output sarà:

```
Inserire il prezzo dell'oggetto: L 2500

Il costo dell'oggetto è L 2500
Le tasse di vendita sono L 202.5
-----------------------------------------
Il costo totale è L 2702.5
```

Potete eseguire questo programma tutte le volte che volete e vi accorgerete che risulterà molto più semplice e veloce di una calcolatrice.

Questo programma dimostra anche che si possono usare variabili e costanti per creare nuove variabili. Notate queste due righe:

```
tassa! = prezzo! * TASSO!   ' Calcola l'ammontare della tassa
totale! = prezzo! + tassa!  ' Calcola il costo totale
```

Come potete vedere queste espressioni contengono solamente variabili numeriche e costanti. Siccome QBasic tratta le variabili numeriche come se fossero numeri, esse possono essere usate ovunque si possa usare normalmente un numero.

SOMMARIO

In questo capitolo abbiamo visto alcuni concetti fondamentali per poter imparare a programmare in QBasic: le variabili (stringa e numeriche) e gli operatori sono infatti elementi indispensabili per la creazione di programmi. Nel prossimo capitolo cercheremo di dotare i programmi di un po' di "intelligenza", cioè della possibilità di prendere decisioni in base a regole prefissate.

DOMANDE ED ESERCIZI

1. Quali sono i quattro tipi di variabili numeriche e in che cosa differiscono?
2. Perché QBasic inserisce uno spazio davanti ai numeri positivi?
3. Che cosa significa quando QBasic mostra un finestra di dialogo con il messaggio "Era atteso: variabile"?
4. Che cosa significa quando QBasic mostra una finestra di dialogo o un messaggio di errore "Overflow"?
5. Che tipo di variabile usereste per ciascuno di questi numeri?

```
a. -32679        d. 14.000001     g. 268110
b. 12.3774       e. -1286.0       h. -10.222222
c. 142286.9      f. -268.0005     i. -0.0000001
```

6. Che differenza c'è tra la divisione regolare e l'operatore /, tra la divisione intera e l'operatore \ , tra la divisione con resto e l'operatore MOD?
7. In ordine di priorità qual è l'ordine di precedenza che QBasic applica agli operatori matematici?
8. Qual è il risultato di questo calcolo?

```
(((5 + 8) - (1 + 3) / 4) * ((7-2) ^ 2))
```

9. Scrivete un programma (completo di commenti) per calcolare i seguenti valori:

```
ABS(-10) + 5
SQR(36)
```

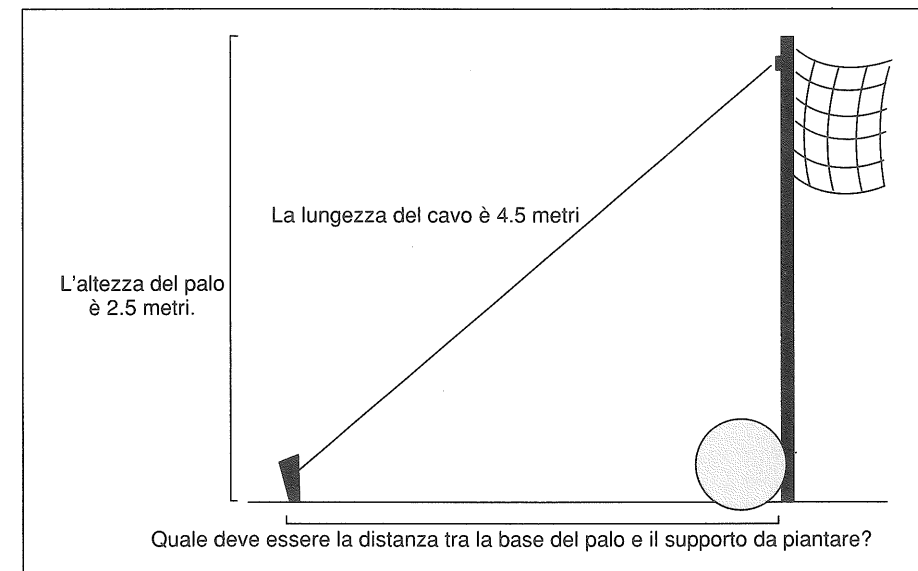


```
SQR(4) ^ 2
COS(3.141592654)
```

10. Il valore della costante matematica pi greco è circa 3.141592654. La formula per il calcolo della circonferenza è:

```
2 * π * raggio
```

Scrivete un programma che chieda all'utente la misura del raggio del cerchio e restituisca la circonferenza.

11. (FACOLTATIVA) Volete mettere su una rete da pallavolo. Supponiamo che l'altezza del palo da terra sia di 2.5 metri e che il cavo attaccato alla cima del palo sia lungo 4.5 metri:

La formula da usare per calcolare la distanza dalla base del palo al supporto dovrebbe essere

$\sqrt{\texttt{lunghezza(cavo)}^2 - \texttt{altezza}^2}$

Scrivete un programma che chiede all'utente di inserire l'altezza del palo e la lunghezza del cavo per restituire, come risultato, la distanza tra la base del palo e il supporto da piantare nel terreno per sostenerlo.

CAPITOLO 5

Controllare il flusso del programma

I programmi scritti fino ad ora sono tutti eseguibili in maniera molto diretta: QBasic passa cioè progressivamente dalla prima all'ultima istruzione. A volte, tuttavia, può capitare di voler eseguire certe porzioni del programma solo in presenza di determinate circostanze. Questo capitolo mostra come avviene questo tipo di programmazione.

INTRODUZIONE AL PROCESSO DECISIONALE

Ogni giorno ciascuno di noi prende decisioni; alcune richiedono la valutazione di una serie di opzioni, altre si prendono quasi senza pensarci.

Alcune decisioni possono condurre a un insieme di azioni; per esempio se state facendo un lungo viaggio in macchina dovete assicurarvi di non restare senza benzina. Proviamo dunque ad esaminare un tipico processo decisionale, come quello che fareste ogni volta che controllate l'indicatore del livello del carburante.

Come mostra il grafico nella pagina successiva, per prima cosa controllate la quantità di benzina nel serbatoio: se è pieno continuate a guidare, altrimenti vi fermate a una stazione per fare il pieno. Allo stesso modo decidete se farvi pulire il parabrezza oppure no: se è pulito pagate e riprendete il vostro viaggio, se è sporco chiedete all'addetto di pulirvelo prima di pagare e ripartire. È facile capire come una sola decisione, basata sulla risposta alla domanda "Ho abbastanza benzina?", vi porti a scegliere di fare o meno una certa altra cosa e a prendere un'ulteriore decisione se necessario.

Processo decisionale in QBasic

QBasic permette di inserire nel programma dei punti decisionali; si può cioè inserire un enunciato "interrogativo" completo di istruzioni che dicono a QBasic che cosa fare a seconda della risposta ottenuta. Quando QBasic incontra una di queste domande, ne determina la risposta: se la risposta è sì, viene attivato un certo gruppo di azioni, se è no un altro.

QBasic agisce sempre in base alle istruzioni date e ammette solo due risposte alle domande: sì e no.

Condizioni Vero e Falso

In un programma in QBasic le domande vengono poste attraverso espressioni condizionali e QBasic valuta queste espressioni non per determinare una risposta del tipo "sì o no" ma per vedere se sono Vere o False (True o False).

Se (IF) l'espressione è condizionale allora (THEN) è Booleana

In QBasic, le espressioni condizionali sono in realtà espressioni booleane (dal nome del matematico inglese del diciannovesimo secolo, George Boole): si dicono booleane le espressioni che possono essere valutate come vere o false. Ecco alcuni esempi di espressioni di questo tipo:

Espressione booleana	Valutazione
Un litro è più grande di un ettolitro	Falso
Dodici è maggiore di dieci	Vero
Cinque è minore o uguale a sei	Vero
Novanta centimetri sono uguali a un metro	Falso

Creazione di espressioni condizionali

Per creare espressioni condizionali (booleane) in un programma bisogna usare un operatore relazionale, un operatore logico o una loro combinazione. QBasic mette a disposizione i seguenti operatori relazionali:

Operatore relazionale	Significato
=	Uguale a
<>	Non uguale a
>	Maggiore di
<	Minore di
>=	Maggiore o uguale a
<=	Minore o uguale a

Questa tabella mostra alcuni esempi di espressioni condizionali che usano operatori relazionali e i loro risultati:

Condizione	Risultato
3 < 7	Vero (3 è inferiore a 7)
14 >= 22	Falso (14 non è maggiore o uguale a 22)
11 <> 16	Vero (11 non è uguale a 16)
5 = –5	Falso (5 non è uguale a –5)
12 > 14	Falso (12 non è maggiore di 14)
11 >= 11	Vero (11 è maggiore o uguale a 11)
totale% < 5	Vero se il valore di totale% è minore a 5; falso negli altri casi
num1% = num2%	Vero se il valore di num1% è uguale al valore di num2%; falso negli altri casi

Fra un po' vedremo alcuni esercizi che utilizzano operatori relazionali e espressioni condizionali, prima però diamo un'occhiata agli enunciati di QBasic che permettono di usare espressioni condizionali nei programmi.

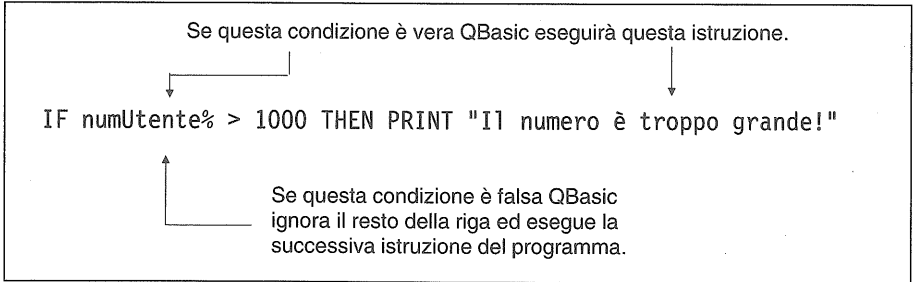
Espressioni numeriche e condizionali

Nel capitolo precedente abbiamo visto le espressioni numeriche. Queste sono molto simili alle espressioni condizionali: entrambe consistono di un operatore e di un'informazione. La differenza consiste nel fatto che un'espressione numerica usa un operatore numerico (per esempio +, –, /, MOD etc.) mentre un'espressione condizionale usa un operatore condizionale (come >=, <, <>, <= etc.). Inoltre la prima restituisce un risultato numerico, la seconda un risultato del tipo "vero o falso".

L'ENUNCIATO IF

L'enunciato IF permette di valutare una condizione e, insieme con la clausola THEN, di prendere un certo corso di azione in base alla valutazione effettuata. Nella forma più semplice IF e THEN formano un solo enunciato, con la seguente sintassi:

```
IF condizione THEN enunciato
```


La condizione è una delle espressioni condizionali appena viste; l'enunciato è un'altra istruzione di QBasic che viene eseguita solo se la condizione è vera; se la condizione è falsa QBasic ignora l'enunciato collegato e passa alla riga successiva del programma. Ecco un esempio:

L'enunciato dopo THEN può essere un qualsiasi enunciato di QBasic ma ricordate che viene eseguito soltanto se la condizione che lo precede è vera.

Conservazione del programma di esercizio

Da questo punto in poi i programmi di esercizio verranno racchiusi in riquadri e riceveranno un nome. Vi raccomandiamo di scrivere questi programmi per conto vostro, così da potervi esercitare sui concetti di programmazione a mano a mano che li presenteremo. Può anche essere utile conservare i programmi di esercizio per creare una raccolta a cui fare riferimento in futuro.

Esercizio: uso dell'enunciato IF

1. Scrivete ed eseguite il programma IF-1.BAS (figura 5-1):
2. Se alla domanda rispondete 11, l'output del vostro programma sarà questo:

```
Quanti giocatori formano una squadra di calcio? 11
Esatto!!
Una squadra di calcio è formata da 11 giocatori
```

Se scrivete 11 la condizione del primo enunciato IF è vera e perciò QBa-

```
' IF-1.BAS
' Questo programma mostra l'uso dell'enunciato IF

CLS

INPUT "Quanti giocatori formano una squadra di calcio? ",
domanda%
IF domanda% = 11 THEN PRINT "Esatto!!"
IF domanda% <>11 THEN PRINT "Sbagliato! La risposta è 11!"

PRINT "Una squadra di calcio è formata da 11 giocatori"
```

Figura 5-1. *IF-1.BAS: un programma che mostra l'uso dell'enunciato IF.*

sic esegue l'enunciato PRINT alla fine della riga. Invece, la seconda condizione IF è falsa e QBasic non esegue il relativo enunciato PRINT.

3. Eseguite il programma inserendo un numero diverso da 11; in questo caso otterrete il seguente risultato:

```
Quanti giocatori formano una squadra di calcio? 13
Sbagliato! La risposta è 11!
Una squadra di calcio è formata da 11 giocatori
```

Questa volta il numero digitato causa un diverso corso di azione; siccome 13 non è uguale a 11 la prima condizione IF è falsa e QBasic non esegue l'enunciato THEN successivo. Si verifica invece la seconda condizione IF e perciò QBasic esegue l'enunciato THEN.

Si noti che in entrambi i casi QBasic mostra l'ultimo enunciato PRINT ("Una squadra di calcio è formata da 11 giocatori"), il che dimostra che anche se la condizione in un enunciato IF è falsa QBasic esegue il resto del programma normalmente; questo avviene perché se la condizione non è vera, solo la parte che segue THEN viene ignorata.

Uso di più condizioni con IF

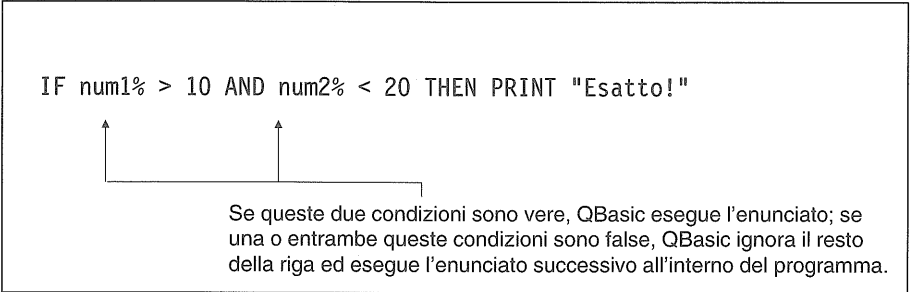
Nel programma precedente QBasic conteneva una sola condizione in ogni enunciato IF. In realtà è possibile inserire più condizioni in un enunciato IF

attraverso gli operatori AND e OR; questi operatori logici si possono usare in espressioni condizionali allo stesso modo in cui gli operatori matematici vengono usati in espressioni numeriche.

L'operatore logico AND

L'operatore logico AND consente di specificare un gruppo di condizioni che devono verificarsi affinché una certa azione venga intrapresa. Questa è la sintassi di un enunciato IF che usa l'operatore AND:

IF *condizione1* AND *condizione2* THEN *enunciato*

Sia la *condizione1* che la *condizione2* devono essere vere perché l'enunciato possa essere eseguito. Eccovi un esempio:

Si noti che l'enunciato IF contiene due condizioni che essendo collegate dall'operatore AND devono essere entrambe vere (num1% maggiore di 10 e num2% minore di 20) affinché QBasic possa eseguire l'enunciato PRINT che segue il THEN.

Esercizio: Uso dell'operatore AND

1. Scrivete ed eseguite il programma IF-2.BAS (figura 5-2)

2. Scrivete 10 come risposta e otterrete questo risultato:

```
Quante persone può contenere una cabina telefonica? 10

Esatto!!
A seconda della loro corporatura, da 10 a 12
persone possono stare in una cabina telefonica.
```

```
' IF-2.BAS
' Questo programma dimostra l'uso dell'operatore logico AND.

CLS

INPUT "Quante persone può contenere una cabina telefonica? ",
indovina%
PRINT
IF indovina% > 9 AND indovina% < 13 THEN PRINT "Esatto!!"
PRINT "A seconda della loro corporatura, da 10 a 12"
PRINT "persone possono stare in una cabina telefonica."
```

Figura 5-2. *Un programma che mostra l'uso dell'operatore logico AND.*

10 è una risposta che verifica entrambe le condizioni contenute nell'enunciato IF. Se provate a eseguire nuovamente il programma e a dare come risposta un valore inferiore a 10 o superiore a 12 una delle due condizioni sarà falsa e QBasic non eseguirà l'enunciato PRINT collegato.

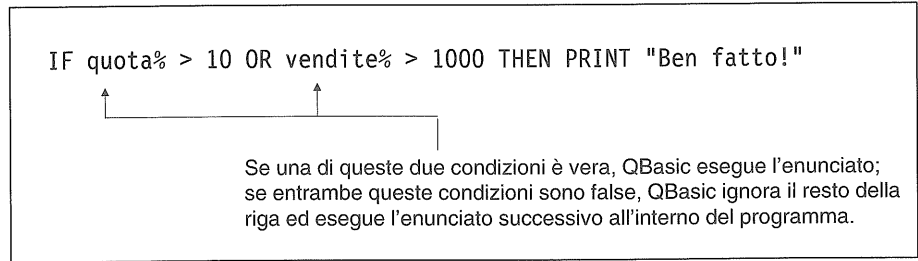
L'operatore logico OR

L'operatore logico OR permette di creare un insieme di condizioni più flessibili che si devono verificare affinché una data azione venga eseguita. La sintassi di un enunciato IF contenente l'operatore OR appare nel modo seguente:

```
IF condizione1 OR condizione2 THEN enunciato
```

Si noti che l'enunciato IF contiene due condizioni di cui soltanto una deve verificarsi prima che QBasic possa eseguire l'enunciato che segue THEN. Naturalmente questa viene eseguita anche nel caso in cui entrambe le condizioni siano vere. Eccovi un esempio:

```
IF quota% > 10 OR vendite% > 1000 THEN PRINT "Ben fatto!"
```

Se una di queste due condizioni è vera, QBasic esegue l'enunciato; se entrambe queste condizioni sono false, QBasic ignora il resto della riga ed esegue l'enunciato successivo all'interno del programma.

Esercizio: Uso dell'operatore OR

1. Scrivete ed eseguite il programma IF-3.BAS (figura 5-3).

```
' IF-3.BAS
' Questo programma mostra l'uso dell'operatore OR

CLS

PRINT "Sto pensando a un numero tra 1 e 100."
PRINT "Puoi indovinare che numero è?"
INPUT "Scrivi un numero tra 1 e 100: ", indovina%
PRINT
IF indovina% = 63 THEN PRINT "Esatto!"
IF indovina% < 53 OR indovina% > 73 THEN PRINT " Sbagliato!"
PRINT "Grazie per aver giocato!"
```

Figura 5-3. *Un programma che dimostra l'uso dell'operatore logico OR.*

2. Eseguite il programma. Scrivete il numero 63 a cui il programma "sta pensando" e lo schermo output apparirà in questo modo:

```
Sto pensando a un numero tra 1 e 100.
Puoi indovinare che numero è?
Scrivi un numero tra 1 e 100: 63

Esatto!
Grazie per aver giocato!
```

3. Eseguite il programma nuovamente usando un numero inferiore a 53 o superiore a 73 e osservate il risultato:

```
Sto pensando a un numero tra 1 e 100.
Puoi indovinare che numero è?
Scrivi un numero tra 1 e 100: 76

Sbagliato!
Grazie per aver giocato!
```

Osservate il secondo enunciato IF: inserendo un numero superiore o inferiore di 11 unità a quello a cui il programma "sta pensando" si ottiene il messaggio "Sbagliato!"

L'operatore logico NOT

L'operatore logico NOT permette di negare una condizione: se una condizione è falsa la rende vera, se è vera la rende falsa. La sintassi di questo operatore è strutturata in questo modo:

`IF NOT` *`condizione`* `THEN` *`enunciato`*

NOT è utile per eseguire un enunciato quando una data condizione non è vera. Osservate questo esempio:

Uso di ELSE con IF e THEN

Abbiamo visto come far in modo che QBasic valuti una condizione ed esegua una certa azione se la condizione è vera o falsa. Che cosa succede se si vuole che QBasic scelga tra due azioni basate su una data condizione?

L'operatore ELSE usato con IF e THEN permette di indicare due azioni separate: una (quella che segue THEN) da eseguire se la condizione è vera, l'altra (quella che segue ELSE) da eseguire se la condizione è falsa.

La sintassi di un enunciato IF usato con THEN e ELSE appare in questo modo:

`IF` *`condizione`* `THEN` *`enunciato1`* `ELSE` *`enunciato2`*

Per condizione si intende la condizione logica che si vuole che QBasic valuti, *enunciato1* è l'enunciato eseguito se la condizione è vera e *enunciato2* è l'enunciato eseguito se la condizione è falsa. IF, THEN e ELSE devono essere tutti contenuti sulla stessa riga:

Esercizio: Uso di IF, THEN e ELSE

1. Scrivete ed eseguite il programma IF-4.BAS (figura 5-4).

```
' IF-4.BAS
' Questo programma dimostra l'uso dell'enunciato ELSE

CLS

PRINT "Sto pensando a un numero tra 1 e 5."
PRINT "Puoi indovinare che numero è?"
INPUT "Scrivi un numero tra 1 e 5: ", indovina%
PRINT
IF indovina% = 3 THEN PRINT "Esatto!" ELSE PRINT "Sbagliato!"
PRINT "Grazie per aver giocato!"
```

Figura 5-4. *Un programma che mostra l'uso della clausola ELSE.*

2. Eseguite il programma inserendo il numero 3 e avrete questo risultato:

```
Sto pensando a un numero tra 1 e 5.
Puoi indovinare che numero è?
Scrivi un numero tra 1 e 5: 3

Esatto!
Grazie per aver giocato!
```

Il numero 3 verifica la condizione e QBasic esegue l'enunciato PRINT che segue THEN.

3. Eseguite il programma inserendo un numero diverso da 3 e otterrete invece quest'altro risultato:

```
Sto pensando a un numero tra 1 e 5.
Puoi indovinare che numero è?
Scrivi un numero tra 1 e 5: 4

Sbagliato!
Grazie per aver giocato!
```

Questa volta la condizione non è verificata e QBasic esegue l'enunciato che segue ELSE.

Creazione di enunciati condizionali più lunghi usando END IF

L'uso di ELSE assieme a IF e THEN permette a QBasic di scegliere tra due opzioni sulla base della valutazione di una certa condizione. Tuttavia, il fatto che queste informazioni debbano essere racchiuse in una sola riga non lascia molto spazio alla creatività.

In questo caso si può usare l'enunciato END IF, che permette di creare programmi più lunghi, collocando azioni diverse su righe separate. La sintassi dell'enunciato IF usata con THEN, ELSE e END appare nel modo seguente:

```
IF condizione THEN
     enunciati eseguiti se la condizione è vera
ELSE
     enunciati eseguiti se la condizione è falsa
ENDIF
```

Questo tipo di enunciato IF consiste di più righe singole, dette blocco. La condizione è l'espressione condizionale che QBasic deve valutare; la parola chiave THEN chiude la riga (si noti che THEN deve sempre trovarsi sulla stessa riga di IF). Se la condizione è vera QBasic:

1. Esegue gli enunciati tra THEN e ELSE.
2. Ignora gli enunciati tra ELSE e END IF.
3. Continua a eseguire il programma.

Tra THEN e ELSE è possibile inserire un numero qualsiasi di enunciati.

Se la condizione è falsa QBasic:

1. Ignora gli enunciati tra THEN e ELSE.
2. Esegue gli enunciati tra ELSE e END IF.
3. Continua a eseguire il programma.

Tra ELSE e END IF è possibile inserire un numero qualsiasi di enunciati. Vediamo un esempio:

```
IF scelta$ = "SI" THEN
     PRINT "Si' signore! Questo modello fa proprio per voi"
     PRINT "Veloce, scattante, un vero divertimento!"
ELSE
     PRINT "Forse Le interessa un altro tipo di vettura"
     PRINT "moderna, elegante e comoda come un salotto!"
END IF
```

Si noti che il rientro tipografico degli enunciati correlati permette di dare maggiore chiarezza al programma.

Esercizio: Uso di IF, THEN, ELSE e END IF

1. Scrivete ed eseguite il programma IF-5.BAS (figura 5-5).
2. Eseguite il programma e rispondete 1885, valore che rende vera la condizione che segue IF; il risultato sarà il seguente:

```
Benvenuto al quiz dell'automobile!

In quale anno Karl Friedrich Benz guidò per la prima
volta con successo un'automobile a benzina? 1885

Esatto! Sei proprio un esperto!
Grazie per aver giocato!
```

3. Ora eseguite il programma usando un anno diverso da 1885 e otterrete questo risultato sul vostro schermo output:

```
Benvenuto al quiz dell'automobile!

In quale anno Karl Friedrich Benz guidò per la prima
```

```
' IF-5.BAS
' Questo programma mostra l'uso di un blocco IF

CLS
PRINT "Benvenuto al quiz dell'automobile!"
PRINT
PRINT "In quale anno Karl Friedrich Benz guidò per la prima"
INPUT "volta con successo un'automobile a benzina? ",
indovina%
PRINT
IF indovina% = 1885 THEN
PRINT "Esatto! Sei proprio un esperto!"
ELSE
PRINT "Sbagliato! Accadde a Mannheim, in Germania,"
PRINT "nel 1885 (la brevettò il 29 gennaio 1886)."
END IF
PRINT "Grazie per aver giocato!"
```

Figura 5.5. *Un programma che dimostra l'uso di un blocco IF.*

```
volta con successo un'automobile a benzina? 1875

Sbagliato! Accadde a Mannheim, in Germania,
nel 1885 (la brevettò il 29 gennaio 1886).
Grazie per aver giocato!
```

Come potete vedere l'uso di END IF con IF, THEN e ELSE vi permette di usare interi blocchi di enunciati.

La parola chiave ELSEIF

ELSEIF è simile a ELSE perché fornisce un corso di azione alternativo nel caso in cui la condizione sia falsa; ELSEIF introduce però anche una condizione ulteriore da valutare.
La sintassi di un enunciato IF con ELSEIF è strutturata in questo modo:


```
IF condizione1 THEN
     Enunciati eseguiti se la condizione 1 è vera
ELSEIF condizione2 THEN
     Enunciati eseguiti se la condizione 2 è vera
ELSEIF condizione3 THEN
     Enunciati eseguiti se la condizione 3 è vera
...
ELSE
     Enunciati eseguiti se tutte le condizioni sono false
ENDIF
```

I puntini tra l'ultima riga ELSEIF e ELSE indicano che si possono avere più enunciati ELSEIF seguite da altre condizioni e non c'è limite al numero di enunciati ELSEIF e di condizioni associate che si possono usare.

NOTA: Bisogna sempre usare THEN alla fine di un enunciato ELSEIF, mentre l'uso di ELSE è opzionale (in questo caso è stato incluso per mostrarne la collocazione nel caso in cui lo si voglia includere).

Se la condizione1 è vera QBasic:

1. Esegue gli enunciati contenuti nelle righe seguenti fino a che incontra il primo ELSEIF.
2. Passa all'istruzione END IF.
3. Continua a eseguire il programma.

Se la condizione1 è falsa QBasic passa alla prima istruzione ELSEIF e valuta la condizione2; se questa è vera QBasic:

1. Esegue gli enunciati contenuti nelle righe seguenti fino a incontrare l'istruzione ELSEIF successiva o la clausola ELSE.
2. Passa all'istruzione END IF e continua a eseguire il programma.

Se la condizione associata a ELSEIF è falsa QBasic passa a quella successiva e ne valuta la condizione; questo processo viene ripetuto fino a che trova un enunciato ELSEIF vero o un enunciato ELSE.
Osservate questo esempio:

```
IF scelta% = 1 THEN
     PRINT "Grazie per aver scelto l'opzione 1!"
ELSEIF scelta% = 2 THEN
     PRINT "Grazie per aver scelto l'opzione 2!"
```

```
ELSEIF scelta% = 3 THEN
     PRINT "Grazie per aver scelto l'opzione 3!"
ELSE
     PRINT "Non avete scelto 1, 2 o 3"
END IF
```

Si noti che l'uso di ELSEIF permette di specificare diverse condizioni e diversi corsi di azione possibili.

Esercizio: Uso di ELSEIF

1. Scrivete il programma ELSEIF.BAS (figura 5-6).
2. Eseguite il programma e inserite l'anno 1959; lo schermo output dovrebbe apparire in questo modo:

```
Benvenuti al quiz cinematografico
In quale anno il film Ben Hur ha vinto
l'Oscar come miglior film?

Scrivete un anno compreso tra il 1950 e il 1959: 1959

Esatto! Ben Hur, diretto da William Wyler,
vinse 11 Oscar nel 1959, compreso quello
per il miglior film.
```

3. Eseguite il programma di nuovo ma questa volta inserite un anno diverso; il risultato sarà il seguente:

```
Benvenuti al quiz cinematografico
In quale anno il film Ben Hur ha vinto
l'Oscar come miglior film?

Scrivete un anno compreso tra il 1950 e il 1959: 1950

Sbagliato! Nel 1950 l'Oscar per il miglior film è stato
vinto dal film Eva contro Eva.
```

4. Infine eseguite il programma inserendo un anno non compreso nell'intervallo tra il 1950 e il 1959; il risultato che otterrete sarà il seguente:


```
' ELSEIF.BAS
' Questo programma illustra l'uso di ELSEIF.
CLS

PRINT "Benvenuti al quiz cinematografico!"
PRINT "In quale anno il film Ben Hur ha vinto"
PRINT "l'Oscar come miglior film?"
PRINT
INPUT "Scrivete un anno compreso tra il 1950 e il 1959: ", anno%
PRINT
IF anno% = 1950 THEN
PRINT "Sbagliato! Nel 1950 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Eva contro Eva."
IF anno% = 1951 THEN
PRINT "Sbagliato! Nel 1951 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Un americano a Parigi."
IF anno% = 1952 THEN
PRINT "Sbagliato! Nel 1952 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Il più grande spettacolo del mondo."
IF anno% = 1953 THEN
PRINT "Sbagliato! Nel 1953 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Da qui all'eternità."
IF anno% = 1954 THEN
PRINT "Sbagliato! Nel 1954 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Fronte del porto."
IF anno% = 1955 THEN
PRINT "Sbagliato! Nel 1955 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Marty."
ELSEIF anno% = 1956 THEN
PRINT "Sbagliato! Nel 1956 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Il giro del mondo in 80 giorni."
ELSEIF anno% = 1957 THEN
PRINT "Sbagliato! Nel 1957 l'Oscar per il miglior film è stato"
PRINT "vinto dal film Il ponte sul fiume Kwai."
ELSEIF anno% = 1958 THEN
PRINT "Sbagliato! Nel 1958 l'Oscar per il miglior film è stato"
PRINT "vinto dala film Gigi."
ELSEIF anno% = 1959 THEN
```

Figura 5-6. *Un quiz cinematografico che usa ELSEIF (continua).*

```
PRINT "Esatto! Ben Hur, diretto da William Wyler,"
PRINT "vinse 11 Oscar nel 1959, compreso quello"
PRINT "per il miglior film."
ELSE
PRINT "Il numero inserito non è compreso tra il 1950 e il 1959."
PRINT "Eseguite il programma di nuovo e inserite un
PRINT "numero appropriato!"
ENDIF
```

Figura 5-6. *Un quiz cinematografico che usa ELSEIF.*

```
Benvenuti al quiz cinematografico
In quale anno il film Ben Hur ha vinto
l'Oscar come miglior film?

Scrivete un anno compreso tra il 1950 e il 1959: 1980

Il numero inserito non è compreso tra il 1950 e il 1959.
Eseguite il programma di nuovo e inserite un
numero appropriato!
```

L'ENUNCIATO SELECT CASE

Un altro strumento per lavorare con enunciati condizionali è l'enunciato SELECT CASE, che svolge una funzione molto simile a quella di IF. In molte occasioni questi due enunciati possono essere usati indifferentemente, ma in realtà ciascuno di essi si adatta meglio a determinate circostanze.

Le parole chiave CASE e END SELECT

Anche l'enunciato SELECT CASE, come IF, deve essere usato in congiunzione con alcune parole chiave, specificatamente CASE e END SELECT.

Di seguito appare la sintassi di SELECT CASE:

```
SELECT CASE variabile
  CASE valore1
    enunciati da eseguire se il valore1 e la variabile sono uguali
  CASE valore2
```



```
    enunciati da eseguire se il valore2 e la variabile sono uguali
  CASE valore3
    enunciati da eseguire se il valore3 e la variabile sono uguali
  ...
END SELECT
```

A differenza di IF, un enunciato SELECT CASE completo non può essere contenuto in una sola riga ma deve essere strutturato in un blocco.

variabile può essere una variabile numerica o una stringa e può essere vista come la porta per le clausole CASE (QBasic usa il valore di *variabile* per determinare quale clausola CASE usare).

Le clausole CASE sono enunciati condizionali separati, il cui *valore* è correlato a *variabile*. Si noti che non c'è limite al numero di singole clausole CASE che si possono inserire tra SELECT CASE e END SELECT.

Ogni clausola CASE è seguita da uno o più enunciati che QBasic esegue se il valore della clausola e quello della variabile che segue SELECT CASE sono uguali. Ecco un esempio:

```
SELECT CASE indovina%
      CASE 1
            PRINT "Spiacente, 1 non è il numero esatto."
      CASE 2
            PRINT "Spiacente, 2 non è il numero esatto."
      CASE 3
            PRINT "Esatto! 3 è il numero esatto!"
      CASE 4
            PRINT "Spiacente, 4 non è il numero esatto."
END SELECT
```

Quando QBasic si trova di fronte un enunciato SELECT CASE, prende nota del valore della variabile (in questo esempio il valore di *indovina%*) e poi esamina il valore specificato nella prima clausola CASE.

Se questo valore è uguale a quello della variabile, QBasic:

1. Esegue gli enunciati che seguono CASE fino a che incontra la clausola CASE successiva o l'enunciato END SELECT.
2. Passa all'enunciato che segue END SELECT e continua con l'esecuzione del programma.

Se il valore della prima clausola CASE e della variabile non sono uguali, QBasic controlla il valore di ogni clausola CASE successiva finché incontra quella giusta.

Se nessuna delle clausole CASE è uguale al valore della variabile contenuta nell'enunciato SELECT CASE, QBasic passa all'enunciato che segue END SELECT e continua con l'esecuzione del programma.

Esercizio: Uso dell'enunciato SELECT CASE

1. Scrivete il programma SELECT-1.BAS (figura 5-7).

```
' SELECT-1.BAS
' Questo programma chiede all'utente di scrivere il suo nome,
' stampa un menu e chiede all'utente di scegliere un elemento.
' Il programma mostra poi un messaggio che varia in base
' all'elemento scelto dal menu.
CLS
INPUT "Scrivete il vostro nome: ", nomeUtente$
PRINT
PRINT "Congratulazioni, "; nomeUtente$; "! Hai vinto la finale"
PRINT "del gioco dei menu!!! Puoi prendere un premio"
PRINT "a vostra scelta:"
PRINT
PRINT , "1. Porta Numero 1"
PRINT , "2. Ciò che si trova dietro la tenda"
PRINT , "3. La grande scatola rosa"
PRINT
INPUT "Scrivete 1, 2 o 3: ", nomeMenu%
PRINT

SELECT CASE nomeMenu
      CASE1
            PRINT "** E' una macchina!!! **"
            PRINT "Si', una nuova Alfa Romeo Spider"
            PRINT "rossa, decappottabile!!"
      CASE2
            PRINT "** E' un buono spesa!!! **"
            PRINT "Cinque milioni da spendere come"
            PRINT "volete, per voi e la famiglia!!"
```

Figura 5-7. *Un programma che mostra l'uso di SELECT CASE (continua).*


```
      CASE3
            PRINT "** E' una scorta di Nutella!! **"
            PRINT "100 chili di Nutella da spalmare"
            PRINT "per tutto l'anno!!!"
END SELECT

PRINT
PRINT "Grazie per aver giocato!"
```

Figura 5-7. *Un programma che mostra l'uso dell'enunciato SELECT CASE.*

2. Eseguite il programma e otterrete un risultato simile a questo:

```
Scrivete il vostro nome: Giovanni

Congratulazioni, Giovanni! avete vinto la finale
del gioco dei menu!!! Potete prendere un premio
a vostra scelta:

1. Porta Numero 1
2. Ciò che si trova dietro la tenda
3. La grande scatola rosa

Scrivete 1, 2 o 3: 2

** E' un buono spesa!!! **
Cinque milioni da spendere come
volete, per voi e la famiglia!!

Grazie per aver giocato!
```

QBasic usa il valore di nomeMenu% per controllare il valore di ciascuna delle clausole CASE. Poiché è stato inserito il numero 2, QBasic ha trovato corrispondenza nel valore della seconda clausola CASE ed ha eseguito gli enunciati associati con quella clausola. Un volta trovata corrispondenza in una delle clausole CASE, QBasic salta quelle rimanenti per eseguire il resto del programma.

3. Eseguite il programma di nuovo usando un valore diverso da 1, 2 o 3 e otterrete questo risultato:

```
Scrivete il vostro nome: Andrea

Congratulazioni, Andrea! avete vinto la finale
del gioco dei menu!!! Potete prendere un premio
a vostra scelta:

1. Porta Numero 1
2. Ciò che si trova dietro la tenda
3. La grande scatola rosa

Scrivete 1, 2 o 3: 5


Grazie per aver giocato!
```

Questa volta QBasic non mostra alcun messaggio correlato alla vincita di un premio perché non ha trovato alcuna corrispondenza tra il valore inserito e quelli contenuti nelle clausole CASE.

Uso di CASE ELSE

Abbiamo visto che, nel caso dell'enunciato IF, la clausola ELSE permette di specificare una o più azioni da eseguire quando le espressioni condizionali sono false. CASE ELSE svolge la stessa funzione per gli enunciati SELECT CASE.

La sintassi per l'uso di questa clausola è la seguente:

```
SELECT CASE variabile
CASE valore
  enunciati da eseguire se il valore e la variabile sono uguali
...
CASE ELSE
  enunciati da eseguire se il valore e la variabile non sono uguali
END SELECT
```


Esercizio: Uso dell'enunciato CASE ELSE

1. Scrivete il programma SELECT-2.BAS (figura 5-8).

```
' SELECT-2.BAS
' Questo programma mostra l'uso della clausola CASE ELSE.

CLS

PRINT "Benvenuti al quiz sugli Oscar cinematografici!"
PRINT "In quale anno il film Ben Hur ha vinto"
PRINT "l'Oscar come miglior film?"
PRINT
INPUT "Scrivete un anno compreso tra il 1950 e il 1959: ", anno%
PRINT

SELECT CASE anno%
  CASE 1950
    PRINT "Sbagliato! Nel 1950 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Eva contro Eva."
  CASE 1951
    PRINT "Sbagliato! Nel 1951 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Un americano a Parigi."
  CASE1952
    PRINT "Sbagliato! Nel 1952 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Il più grande spettacolo del mondo."
  CASE1953
    PRINT "Sbagliato! Nel 1953 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Da qui all'eternità."
  CASE1954
    PRINT "Sbagliato! Nel 1954 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Fronte del porto."
  CASE1955
    PRINT "Sbagliato! Nel 1955 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Marty."
  CASE1956
    PRINT "Sbagliato! Nel 1956 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Il giro del mondo in 80 giorni."
  CASE1957
    PRINT "Sbagliato! Nel 1957 l'Oscar per il miglior film è stato"
    PRINT "vinto dal film Il ponte sul fiume Kwai.
```

Figura 5-8. *Un quiz cinematografico che usa CASE ELSE (continua).*

```
  CASE1958
   PRINT "Sbagliato! Nel 1958 l'Oscar per il miglior film è stato"
   PRINT "vinto dal film Gigi."
  CASE1959
   PRINT "Esatto! Ben Hur, diretto da William Wyler,"
   PRINT "vinse 11 Oscar nel 1959, compreso quello"
   PRINT "per il miglior film."
  CASE ELSE
   PRINT "Non avete scritto un numero compreso tra 1950 e 1959."
   PRINT "Eseguite il programma di nuovo e inserite"
   PRINT "un numero appropriato!"
END SELECT
```

Figura 5-8. *Un quiz cinematografico che usa CASE ELSE.*

2. Eseguite il programma inserendo l'anno corretto e il vostro schermo apparirà in questo modo:

```
Benvenuti al quiz sugli Oscar cinematografici!
In quale anno il film Ben Hur ha vinto
l'Oscar come miglior film?

Scrivete un anno compreso tra il 1950 e il 1959: 1959

Esatto! Ben Hur, diretto da William Wyler,
vinse 11 Oscar nel 1959, compreso quello
per il miglior film.
```

3. Provate ora a eseguire il programma inserendo un valore non compreso nell'intervallo definito e otterrete questo risultato:

```
Benvenuti al quiz sugli Oscar cinematografici!
In quale anno il film Ben Hur ha vinto
l'Oscar come miglior film?

Scrivete un anno compreso tra il 1950 e il 1959: 1958

Non avete scritto un numero compreso tra 1950 e 1959.
Eseguite il programma di nuovo e inserite
un numero appropriato!
```

Siccome QBasic non ha trovato alcuna clausola CASE uguale alla variabile ha eseguito l'enunciato successivo a CASE ELSE. Perciò l'uso di CASE ELSE consente di gestire situazioni non previste negli enunciati SELECT CASE.

Uso di IS assieme a CASE

La parola chiave IS permette di usare un'*espressione* condizionale (al posto di un valore numerico singolo o di una stringa) all'interno di una clausola CASE, per cui QBasic esegue gli enunciati che seguono quella clausola solo se la condizione è vera. La sintassi di una clausola CASE con IS è la seguente:

```
SELECT CASE variabile
      CASE IS condizione
      enunciati da eseguire se la condizione è vera
      ...
END SELECT
```

Questa sintassi è identica al normale enunciato SELECT CASE, con l'aggiunta della "*condizione* IS" alla clausola CASE. In questo caso QBasic usa il valore delle variabili SELECT CASE come base per il confronto.

Ecco un esempio:

```
SELECT CASE userNum%
      CASE IS <= 4
            PRINT "Il numero immesso è 4 o inferiore"
      CASE 5
            PRINT "Il numero immesso è 5"
      CASE IS >= 6
            PRINT "Il numero imesso è 6 o superiore"
ENDSELECT
```

QBasic confronta il valore digitato con le parti condizionali delle clausole CASE. In questo esempio se l'utente digitasse un valore uguale o inferiore a 4, la condizione della prima clausola CASE sarebbe vera e QBasic eseguirebbe l'enunciato PRINT che gli è associato. Si noti che la seconda clausola CASE usa semplicemente un valore: valori ed espressioni condizionali IS possono essere mescolati.

Esercizio: Uso di SELECT CASE e IS

1. Scrivete il programma SELECT-3.BAS (figura 5-9)

```
' SELECT-3.BAS
' Questo programma analizza il consumo giornaliero
' di caffè dell'utente.

CLS

INPUT "Quanti caffè hai bevuto oggi? ", tazzeCaffe%
PRINT

SELECT CASE tazzeCaffe%
      CASE 0
            PRINT "Non ti piace il mio caffè??"
      CASE IS <= 3      ' da 1 a 3 tazze al giorno
            PRINT "Una quantità moderata."
      CASE IS <= 7      ' da 4 a 7 tazze al giorno
            PRINT "Meglio berlo decaffeinato..."
      CASE IS >= 8      ' 8 o più tazze al giorno
            PRINT "Troppi caffè!!!"
END SELECT
```

Figura 5-9. *Un programma che usa IS per analizzare il consumo di caffè..*

2. Eseguitelo e inserite un valore; il vostro schermo output apparirà in questo modo:

```
Quanti caffè hai bevuto oggi? 3

Una quantità moderata.
```

Premete un tasto per tornare nella Finestra di digitazione e osservate come vengono impostate le condizioni nel programma. Se scrivete 3, come abbiamo fatto noi, risulta verificata la seconda clausola CASE; notate che tecnicamente il valore 3 renderebbe vera anche la terza clausola e quindi ci si potrebbe chiedere perché QBasic non esegua l'enunciato PRINT associato.

A questo proposito ricordate che non appena QBasic trova una clausola CASE che verifica la condizione, salta tutte le altre clausole e continua con l'esecuzione del programma.

Uso di TO con SELECT CASE

Per specificare un intervallo di valori validi per una clausola CASE si usa la parola chiave TO, che presenta la seguente sintassi:

```
SELECT CASE variabile
      CASE valore1 TO valore2
      enunciati da eseguire se la condizione è vera
      ...
END SELECT
```

I due valori nella clausola CASE possono essere numerici o stringhe di testo (entrambe vanno racchiuse tra virgolette).

Uso di valori numerici con la parola chiave TO

Di seguito mostriamo un esempio che usa due valori numerici con la parola chiave TO in una clausola CASE:

```
SELECT CASE userNum%
      CASE 1 TO 5
            PRINT "Il numero immesso era compreso tra 1 e 5"
      CASE 6 TO 10
            PRINT "Il numero immesso era compreso tra 6 e 10"
END SELECT
```

Si noti che il valore più piccolo si trova alla sinistra di TO; i valori vanno sempre posti in quest'ordine, altrimenti QBasic non interpreterà la condizione in maniera corretta e considererà sempre l'espressione come falsa.

Questo vale anche per valori negativi; l'esempio che segue mostra la posizione corretta di un valore negativo:

```
CASE -12 TO 3
```

Siccome –12 è inferiore a 3 va posto alla sinistra di TO.

Uso di stringhe con la parola chiave TO

TO permette di usare anche un intervallo di stringhe all'interno di una clausola CASE, come in questo esempio:

```
SELECT CASE parola$
      CASE "a" TO "m"
            PRINT "La lettera digitata è compresa tra a e m."
      CASE "m" TO "z"
            PRINT "La parola digitata è compresa tra m e z."
END SELECT
```

Anche se a prima vista non sembrerebbe così, un intervallo di stringhe funziona nello stesso modo di un intervallo di numeri.

In un esempio precedente abbiamo visto che QBasic tratta ogni carattere separatamente (per QBasic, cioè, le lettere a ed A sono diverse).

La ragione va trovata nel modo in cui un computer gestisce le informazioni. Per quanto sia possibile digitare lettere maiuscole e minuscole sulla tastiera e vederle sullo schermo, un computer non è in grado di gestire le lettere direttamente. Il microchip interno al computer può lavorare solo con numeri; quando si preme un tasto il computer traduce temporaneamente quel carattere in un numero con cui è in grado di lavorare.

Quando poi è necessario usare quel numero sotto forma di carattere (per esempio visualizzarlo sullo schermo) QBasic lo converte nuovamente nel carattere originariamente digitato.

Naturalmente non c'è bisogno di imparare come avviene la gestione dei numeri, ma è necessario sapere che ciò accade per capire come QBasic gestisce caratteri individuali o stringhe di caratteri. Per cui anche se usate caratteri o stringhe il computer e QBasic li considerano semplicemente come numeri.

Nell'esempio precedente avrete notato la sovrapposizione tra le due clausole CASE: nella prima l'intervallo va da *a* a *m*, nella seconda da *m* a *z*. Perché vi è questa sovrapposizione? L'esercizio seguente ne spiega le ragioni.

Esercizio: Uso di un intervallo di stringhe in una clausola CASE

1. Scrivere il programma TO-1.BAS (figura 5-10).

2. Eseguite il programma e scrivete il nome *carlo*; sul vostro schermo apparirà questo risultato:

```
Scrivete il vostro nome: carlo

Il nome è compreso tra a e m
```



```
' TO-1.BAS
' Questo programma mostra l'uso della parola chiave TO
' in una clausola CASE

CLS

INPUT "Scrivete il vostro nome: ", nome$
PRINT
SELECT CASE nome$
      CASE "a" TO "m"
            PRINT "Il nome è compreso tra a e m"
      CASE "m" TO "z"
            PRINT "Il nome è compreso tra m e z"
      CASE "A" TO "M"
            PRINT "Il nome è compreso tra A e M"
      CASE "M" TO "Z"
            PRINT "Il nome è compreso tra M e Z"
END SELECT
```

Figura 5-10. *Un programma che usa TO per confrontare del testo.*

Siccome la prima lettera è minuscola risulta vera la prima enunciato CASE.

3. Eseguite il programma di nuovo e scrivete solo *m*; il risultato che otterrete sarà il seguente:

```
Scrivete il vostro nome: m

Il nome è compreso tra a e m
```

Anche in questo caso siccome la lettera *m* verifica la prima clausola CASE QBasic esegue l'enunciato correlato a questa clausola.

4. Eseguite il programma scrivendo il nome *marco* e otterrete questo risultato:

```
Scrivete il vostro nome: marco

Il nome è compreso tra m e z
```

Si noti che in questo caso il nome *carlo* verifica la seconda clausola CASE mentre nell'esempio precedente la sola lettera *m* verificava la prima clausola CASE. Perché? Ricordate che in un computer tutte le lettere vengono trattate come numeri; in un certo senso si può immaginare che una stringa di caratteri sia come una serie di numeri. Nell'esempio precedente la parola marco ha un "valore" più alto della lettera *m*, così come 15 ha un valore più alto di 1. Anche se 15 e 1 cominciano con lo stesso numero, le regole matematiche ci dicono che 15 ha un valore più alto di 1.

Questa regola può essere applicata, in generale, anche a stringhe di caratteri. Nell'esempio fatto QBasic non considera l'intera parola *marco* nell'esaminare le clausole CASE; gli basta infatti esaminare le prime due lettere: *ma*. Siccome la combinazione *ma* ha un valore maggiore della semplice lettera m e questa rappresenta l'estremo superiore dell'intervallo da *a* a *m*, QBasic riconosce la prima condizione come non vera.

La seconda condizione risulta invece verificata. Sebbene la stringa *marco* sia più lunga di una sola lettera, la prima lettera di questa parola cade nell'intervallo da *m* a *z* e perciò QBasic determina che la seconda condizione è verificata.

Esercizio: Un altro esempio di un intervallo di stringhe di testo

1. Modificate il programma TO-1.BAS per ottenere TO-2.BAS (figura 5-11). TO-2.BAS è identico a TO-1.BAS tranne che per alcune modifiche alle condizioni — specialmente la prima e la terza clausola CASE.

2. Eseguite il programma e scrivete la lettera maiuscola *M*; a video otterrete il seguente risultato:

```
Scrivete il vostro nome: M

Il nome è compreso tra A a M
```

3. Eseguite il programma di nuovo e scrivete *Marco*; il risultato sarà il seguente:


```
' TO-2.BAS
' Questo programma mostra l'uso di TO in una clausola CASE

CLS
INPUT "Scrivete il vostro nome: ", nome$
PRINT

SELECT CASE nome$
     CASE "a" TO "mz"
          PRINT "Il nome è compreso tra a e m"
     CASE "n" TO "z"
          PRINT "Il nome è compreso tra n e z"
     CASE "A" TO "Mz"
          PRINT "Il nome è compreso tra A e M"
     CASE "N"" TO "Z"
          PRINT "Il nome è compreso tra N e Z"
END SELECT
```

Figura 5-11. *Una versione modificata di TO-1.BAS.*

```
Scrivete il vostro nome: Marco

Il nome è compreso tra A e M
```

Questa volta è la condizione nella terza clausola CASE che risulta verificata, perché *Mz* ha un "valore" maggiore delle prime due lettere di *Marco*.

Uso di condizioni multiple con CASE

Fino ad ora la condizione usata nelle clausole CASE è stata rappresentata da un unico valore, da una condizione o da un ventaglio di valori. QBasic permette di usare le virgole per separare più condizioni all'interno di una clausola CASE. Si consideri il seguente esempio:

```
SELECT CASE numero%
     CASE 1, 3, 5, 7, 9
          PRINT "Il numero immesso era dispari"
     CASE 2, 4, 6, 8, 10
          print "IL NUMERO IMMESSO ERA PARI"
END SELECT
```

Usare una virgola in una clausola CASE è come usare l'operatore OR in un enunciato IF. Se uno degli elementi verifica la clausola CASE QBasic esegue gli enunciati associati.

Questi elementi non devono essere dello stesso tipo e si possono mescolare valori, condizioni e intervalli, come in questo esempio:

```
CASE 5, IS  6, 20 TO 30
```

Esercizio: Uso delle virgole nelle clausole CASE

Scrivete ed eseguite il programma TO-3.BAS (figura 5-12). Come potete vedere l'uso delle virgole per specificare elementi multipli aggiunge notevole flessibilità agli enunciati SELECT CASE.

```
' TO-3.BAS
' Questo programma mostra l'uso delle virgole in clausole CASE

CLS

PRINT "Scrivete un mese e vi dirò quanti giorni"
PRINT "di vacanza ci sono negli USA durante quel mese."
INPUT "Scrivere un numero da 1 a 12: ", mese%
PRINT

SELECT CASE mese%
  CASE 8
    PRINT "Non ci sono vacanze negli USA durante questo mese"
  CASE 3, 4, 7
    PRINT "Un giorno di vacanza negli USA durante questo mese"
  CASE 6
    PRINT "Due giorni di vacanza in USA durante questo mese"
  CASE 1, 2, 9 TO 11
    PRINT "Tre giorni di vacanza in USA durante questo mese"
  CASE 12
    PRINT "Quattro giorni di vacanza in USA durante questo mese"
  CASE 5
    PRINT "Cinque giorni di vacanza in USA durante questo mese"
END SELECT
```

Figura 5-12. *TO-3.BAS. Un programma che dimostra l'uso di condizioni multiple entro enunciato CASE.*

CHE TIPO DI ENUNCIATI CONDIZIONALI È MEGLIO USARE?

Sia IF sia SELECT CASE contengono moltissime opzioni e configurazioni diverse. Rispondere alle domande seguenti può aiutarvi a scegliere l'enunciato condizionale più appropriato da inserire nel vostro programma.

- Con quale tipo di enunciato mi trovo più a mio agio? Non dimenticate che grazie all'uso di parole chiave come ELSE o ELSEIF, per l'enunciato IF e CASE o CASE ELSE, per l'enunciato SELECT CASE, è possibile ottenere gli stessi risultati da entrambe le espressioni. Usate quindi l'enunciato con cui vi sentite più a vostro agio.
- Quante sono le condizioni contenute nel vostro enunciato? SELECT CASE richiede un minimo di tre istruzioni separate per impostare anche una sola condizione! Perciò, se avete soltanto una condizione può essere consigliabile l'uso dell'enunciato IF.
  Se avete due o più condizioni IF e SELECT CASE sono ugualmente facili da usare.

Se avete molte condizioni, SELECT CASE può essere la scelta migliore, perché risulta più semplice da leggere e da capire di un enunciato IF. Un enunciato SELECT CASE va anche meglio in tutti quei casi in cui volete cercare un numero di valori specifici che non ricadono perfettamente in un intervallo.

SOMMARIO

Gli enunciati condizionali rendono i programmi più flessibili e "intelligenti". Essi permettono di controllare l'esecuzione di un programma sulla base delle informazioni digitate dall'utente o modificate dal programma in fase di esecuzione. Le espressioni condizionali diventeranno parte integrante della maggior parte dei vostri programmi.

DOMANDE ED ESERCIZI

1. Che differenze ci sono tra un'espressione numerica e una condizionale?

2. Quali, tra questi, non sono operatori condizionali?

```
a. <=     d. <     g. <>     l. /
b. ><     e. >=    h. >      m. <<
c. ==     f. ^     i. =<     n. =
```

3. Vero o Falso. Bisogna sempre usare THEN sulla stessa riga di IF.
4. Qual è la differenza tra l'operatore AND e l'operatore OR?
5. Qual è la funzione di ELSE? In che cosa si differenzia da THEN?
6. Come funziona la funzione ELSEIF? Quali altre parole chiave vanno usate con ELSEIF?
7. Scrivete un programma che ponga all'utente una domanda a cui è possibile rispondere con un sì o con un no e istruitelo a scrivere S per sì e N per no. Includete un gruppo di enunciati che verranno eseguiti se l'utente digita S e un gruppo se l'utente digita N. Includete un terzo gruppo di enunciati che verranno eseguiti se l'utente non dà una risposta corretta. Un consiglio: non dimenticate di controllare la corrispondenza maiuscole/minuscole.
8. Come funziona l'enunciato SELECT CASE?
9. Che funzione svolge la clausola CASE?
10. Che funzioni svolgono IS e TO in un enunciato CASE?
11. Scrivete un programma che invita l'utente a descrivere tre oggetti. Il programma deve mostrare una lista numerata di oggetti e chiedere all'utente di scrivere il numero corrispondente all'oggetto da descrivere. Usate SELECT CASE per visualizzare le informazioni appropriate e includete degli enunciati da eseguire se l'utente non dà una risposta appropriata.

CAPITOLO 6

Uso dei loop di QBasic

Finora avete appreso la maggior parte dei principi fondamentali del linguaggio QBasic. In questo capitolo impareremo a usare uno degli strumenti più potenti di QBasic: il *loop*. Usando i loop potremo risparmiare molto tempo prezioso facendo eseguire al computer compiti ripetitivi in molto meno tempo.

INTRODUZIONE AI LOOP DI QBASIC

Un *loop* consiste semplicemente nella ripetizione di uno o più enunciati di QBasic. I loop si possono ripetere in due modi:

- Un determinato numero di volte.
- Fino a quando si verifica una certa condizione.

In QBasic, tre enunciati, FOR, WHILE, e DO permettono di aggiungere dei loop al programma. Nell'enunciato FOR, viene specificato per quante volte deve essere eseguito il loop. Negli enunciati WHILE e DO, va specificata una condizione; QBasic la valuta ed esegue il loop fino a quando la condizione non risulta soddisfatta. Sebbene gli enunciati FOR, WHILE, e DO eseguano funzioni simili, ognuno di essi svolge il suo compito in modo leggermente diverso. Come vedremo, le variazioni fanno sì che ogni enunciato sia adatti a un particolare tipo di lavoro.

L'AFFERMAZIONE FOR

Per creare un loop che svolga il compito assegnato uno specifico numero di volte (il tipo di loop più comune in QBasic), si usa l'enunciato FOR. L'enunciato FOR termina sempre con l'enunciato NEXT, come si vede chiaramente dalla sua sintassi:

```
FOR variabile = inizio TO fine
     enunciati che devono essere ripetuti
NEXT variabile
```

- *variabile* è una variabile numerica che indica quante volte QBasic ha eseguito il loop. QBasic incrementa *variabile* ogni volta che esegue il loop. Si noti che *variabile* segue sia l'enunciato FOR che l'enunciato NEXT. I nomi della variabile devono essere esattamente gli stessi.

- *inizio* è il valore numerico, un numero o un'espressione numerica, da cui si vuol far cominciare QBasic a contare.
- *fine* è un valore numerico, un numero o un'espressione numerica, che dice a QBasic quanto deve contare, cioè, quante volte deve ripetere il loop. Quando si assegnano dei valori a *inizio* e *fine*, bisogna tenere a mente i seguenti suggerimenti:
- Per *inizio* e *fine* sono ammessi valori sia positivi che negativi.
- Per *inizio* e *fine* sono ammessi sia numeri interi che valori che contengono numeri decimali.
- Il valore di *inizio* deve essere inferiore o uguale al valore di *fine*.
- Il valore di *inizio* non deve essere necessariamente 1.

Gli enunciati tra FOR e NEXT sono quelli che QBasic esegue il numero di volte specificato. Non c'è limite al numero di enunciati che si possono usare, e inoltre gli enunciati non devono essere necessariamente dello stesso tipo.

QBasic usa l'enunciato NEXT per contare quante volte ha eseguito gli enunciati fra FOR e NEXT. Ecco un esempio di un enunciato FOR che stampa un messaggio cinque volte, incrementando ogni volta la variabile *i%*:

```
FOR i% = 1 TO 5
     PRINT "Sono in un loop."
NEXT i%
```

Tutte le volte che ha completato un loop, QBasic torna di nuovo all'enunciato FOR per confrontare il valore di *i%* con il valore di *fine* (che è 5).

Non appena *i%* supera *fine*, QBasic torna all'enunciato che segue NEXT e continua a eseguire il programma.

Esercizio: Uso del loop di tipo FOR

1. Scrivete il programma FOR-1.BAS (figura 6-1).
2. Eseguite il programma; il risultato dovrebbe essere il seguente:

```
Scrivi un numero da 2 a 5:  4

Queste righe verranno stampate 4 volte.
Questa è la volta 1
```

```
' FOR-1.BAS
' Questo programma mostra l'uso del loop di tipo FOR.

CLS

INPUT "Scrivi un numero da 2 a 5: ", volte%
PRINT

FOR i% = 1 TO volte%
     PRINT "Queste righe verranno stampate"; volte%; "volte."
     PRINT "Questa è la volta"; i%
     PRINT
NEXT i%
```

Figura 6-1. *Un semplice loop di tipo FOR.*

```
Queste righe verranno stampate 4 volte.
Questa è la volta 2

Queste righe verranno stampate 4 volte.
Questa è la volta 3

Queste righe verranno stampate 4 volte.
Questa è la volta 4
```

Esercizio: Quando inizio è maggiore di fine

1. Scrivete il programma FOR-2.BAS (figura 6-2). Osservate che *inizio* è maggiore di *fine*.
2. Eseguite il programma. Vedrete che a eccezione del messaggio "Premi un tasto per continuare", la schermata dell'output apparirà vuota. QBasic infatti ha visto che il valore di *inizio* era 6, un valore maggiore di quello di *fine* e ha ritenuto concluso il suo compito.


```
' FOR-2.BAS
' Questo programma mostra un loop di tipo FOR che non può
essere eseguito.

CLS

FOR i% = 6 TO 5
     PRINT "Il valore corrente di i% è"; i%
NEXT i%
```

Figura 6-2. *Un loop di tipo FOR che non può essere eseguito.*

Perché usare i%

Avrete notato che la maggior parte dei loop FOR in questo libro usano *i%* come variabile di loop. Per quale motivo?

Prima ancora che il BASIC venisse inventato, molti programmatori usavano un linguaggio chiamato FORTRAN. In FORTRAN, la prima lettera di un nome di variabile ne specificava il tipo. Per esempio, un qualunque nome di variabile che cominciasse con una lettera da *I* a *N* denotava una variabile intera, così come ora in QBasic un segno di percentuale (%) aggiunto ad un nome di variabile denota una variabile intera.

La maggior parte dei valori dei loop di tipo FOR sono numeri interi. Così, per risparmiare tempo, un programmatore in FORTRAN di solito usava la variabile *i* come valore di loop. I programmatori che avevano bisogno di nidificare due o più loop (si veda a questo proposito il paragrafo "Nidificare i loop di tipo FOR" più avanti in questo capitolo) usavano *j*, poi *k* e così via. Molti programmatori in BASIC hanno adottato questa terminologia.

È necessario usare *i%*, *j%* e *k%*? No, si può usare un qualsiasi nome di variabile di QBasic (per esempio, *conta%*, *numRighe%* o *giornoMese%*).

Esercizio: Usare valori uguali per inizio e fine

1. Cambiate l'enunciato FOR nel programma FOR-2.BAS in modo che *inizio* e *fine* abbiano lo stesso valore:

```
FOR i% = 5 TO 5
```

2. Eseguite il programma e otterrete questo risultato:

```
Il valore corrente di i% è 5
```

Questa volta QBasic esegue il corpo dell'enunciato FOR una sola volta. Infatti, dal momento che i valori di *inizio* e *fine* sono gli stessi, QBasic dà per scontato che questo sia il suo ultimo "giro" ed esegue l'enunciato PRINT una volta prima di andare avanti.

Il looping e l'enunciato COLOR

Si parlerà più diffusamente dell'enunciato COLOR nel capitolo 11. Nell'esercizio che segue assisteremo a una dimostrazione in cui verrà impiegato questo enunciato insieme a un loop di tipo FOR.

Esercizio: Cambiare i colori di fondo con un loop di tipo FOR

1. Scrivete il programma FOR-3.BAS (figura 6-3).

```
' FOR-3.BAS
' Questo programma usa un loop di tipo FOR e l'enunciato
' COLOR per mostrare l'uso dei vari colori di fondo.

CLS

FOR i% = 1 TO 15 ' Usa i colori da 1 a 15
     COLOR i%    ' Usa il valore di i% per il colore
     PRINT "Questa riga è stampata con il colore"; i%
NEXT i%
```

Figura 6-3. *Uso dell'enunciato COLOR in un loop di tipo FOR.*

NOTA: È possibile eseguire questo programma anche se non si dispone di un monitor a colori.

L'enunciato COLOR

L'enunciato COLOR permette di specificare il testo in primo piano e i colori di fondo per la schermata di output. Ecco la sintassi per l'enunciato COLOR:

```
COLOR [primo piano][, sfondo]
```

Primo piano e *sfondo* sono numeri che rappresentano dei colori:

Valore	Colore risultante
0	Nero (sfondo predefinito)
1	Blu
2	Verde
3	Azzurro
4	Rosso
5	Magenta
6	Marrone
7	Bianco (primo piano predefinito)
8	Grigio
9	Celeste
10	Verde chiaro
11	Azzurro chiaro
12	Rosso chiaro
13	Magenta chiaro
14	Giallo
15	Bianco brillante

Il seguente enunciato:

```
COLOR 14, 1
```

fa sì che QBasic visualizzi un testo giallo su sfondo blu.

2. Eseguite il programma. Lo schermo avrà questo aspetto ma ogni riga sarà di un colore diverso:

```
Questa riga è stampata con il colore 1
Questa riga è stampata con il colore 2
Questa riga è stampata con il colore 3
```

```
Questa riga è stampata con il colore 4
Questa riga è stampata con il colore 5
Questa riga è stampata con il colore 6
Questa riga è stampata con il colore 7
Questa riga è stampata con il colore 8
Questa riga è stampata con il colore 9
Questa riga è stampata con il colore 10
Questa riga è stampata con il colore 11
Questa riga è stampata con il colore 12
Questa riga è stampata con il colore 13
Questa riga è stampata con il colore 14
Questa riga è stampata con il colore 15
```

Esercizio: Cambiare i colori di primo piano e di fondo con un loop di tipo FOR

1. Scrivete il programma FOR-4.BAS (figura 6-4).

```
' FOR-4.BAS
' Questo programma usa un loop di tipo FOR e l'enunciato
' COLOR per cambiare i colori di primo piano e di fondo.

CLS

FOR i% = 1 TO 15 ' Usa i colori da 1 a 15
     COLOR i%, i% - 1
     PRINT "Il colore corrente in primo piano è"; i%;
     PRINT "; il colore di fondo è"; i% - 1
NEXT i%
```

Figura 6-4. *Cambiare i colori di primo piano e di fondo con un loop di tipo FOR.*

2. Eseguite il programma; il testo che apparirà a video sarà a colori con diversi colori di fondo e primo piano:

```
Il colore di primo piano è 1; il colore di fondo è 0
Il colore di primo piano è 2; il colore di fondo è 1
Il colore di primo piano è 3; il colore di fondo è 2
Il colore di primo piano è 4; il colore di fondo è 3
Il colore di primo piano è 5; il colore di fondo è 4
Il colore di primo piano è 6; il colore di fondo è 5
```



```
Il colore di primo piano è 7; il colore di fondo è 6
Il colore di primo piano è 8; il colore di fondo è 7
Il colore di primo piano è 9; il colore di fondo è 8
Il colore di primo piano è 10; il colore di fondo è 9
Il colore di primo piano è 11; il colore di fondo è 10
Il colore di primo piano è 12; il colore di fondo è 11
Il colore di primo piano è 13; il colore di fondo è 12
Il colore di primo piano è 14; il colore di fondo è 13
Il colore di primo piano è 15; il colore di fondo è 14
```

3. Premete un tasto per tornare alla Finestra di digitazione e osservate l'enunciato COLOR contenuta nel loop FOR. Guardate attentamente i valori che definiscono i colori di primo piano e di fondo: il colore di primo piano è definito dal valore di *i%* mentre il colore di fondo è definito dal valore di *i%* meno 1. In questo modo si evita di definire con lo stesso valore il colore di primo piano e il colore di fondo. Se questi valori fossero gli stessi non si potrebbe più leggere il testo.

Il looping e l'enunciato SOUND

Ancora una volta si parlerà più diffusamente dell'enunciato SOUND nel capitolo 11. Nell'esercizio che segue, l'enunciato FOR unito all'enunciato SOUND fornisce una chiara prova della versatilità dei loop.

L'enunciato SOUND

L'enunciato SOUND fa sì che il computer crei un suono. La sintassi dell'enunciato SOUND è la seguente:

```
SOUND frequenza, durata
```

L'argomento *frequenza* è un numero intero da 37 a 32.767 che indica la frequenza del suono in cicli per secondo, o hertz. L'argomento *durata* è un numero intero o un numero che contiene decimali da 0 a 65.535 che indica quanto debba durare il suono. (Il valore 18.2 è uguale a un secondo.) Per esempio, questo enunciato fa sì che QBasic produca un suono di 500 hertz per mezzo secondo:

```
SOUND 500, 9.1
```

Esercizio: Uso delle enunciati FOR e SOUND

1. Scrivete il programma FOR-5.BAS (figura 6-5).

```
' FOR-5.BAS
' Questo programma usa l'enunciato SOUND in un loop di
' tipo FOR per produrre degli effetti sonori.

CLS

PRINT , "GENERATORE DI EFFETTI SONORI"
PRINT , "Scegli il suono che vorresti sentire"
PRINT , "dal seguente menu:"
PRINT
PRINT , "1. Pistola a razzo"
PRINT , "2. Sirena"
PRINT , "3. Decollo!"
PRINT

INPUT "digita 1, 2, o 3:  ", scelta%
PRINT
SELECT CASE scelta%
     CASE 1            ' Pistola a razzo
          FOR i% = 1 TO 50
               SOUND 850, .3
               SOUND 800, .3
               SOUND 825, .3
          NEXT i%
     CASE 2            ' Sirena
          FOR i% = 1 TO 5
               SOUND 500, 10
               SOUND 450, 10
          NEXT i%
     CASE 3            ' Decollo!
          FOR i% = 500 TO 2500
               SOUND i%, .03     ' Usa il valore di i% per
                                   il suono
          NEXT i%
END SELECT
```

Figura 6-5. *FOR-5.BAS: un programma per la generazione dei suoni.*

2. Eseguite il programma; a video apparirà questo invito:

```
GENERATORE DI EFFETTI SONORI
Scegli il suono che vorresti sentire
dal seguente menu:

1. Pistola a razzo
2. Sirena
3. Decollo!

Digita 1, 2, o 3:
```

3. Digitate il valore appropriato per sentire il suono che avete scelto.

Eseguite il programma alcune volte per provare ogni suono, dopodiché sperimentate questo programma cambiando i valori nei vari enunciati SOUND ed eseguite nuovamente il programma. Noterete che la maggior parte dei valori di durata sono bassi; infatti, i valori di durata più bassi vanno meglio per gli effetti sonori, mentre quelli più alti verranno usati per creare una melodia.

Controllare il calcolo con STEP

Come si è visto, quando QBasic esegue un loop di tipo FOR comincia con l'assegnare il valore di *inizio* alla variabile che segue FOR e poi esegue il loop attraverso il blocco degli enunciati sino a che il valore della variabile contatore è più grande del valore di *fine*. Ogni volta che incontra l'enunciato NEXT, QBasic incrementa il valore del contatore di 1.

Servendosi della clausola STEP si può anche dire a QBasic di incrementare il contatore di un valore diverso da 1, come avviene in questo esempio:

```
FOR i% = 5 TO 25 STEP 5
     PRINT "Questo messaggio verrà stampato 5 volte."
NEXT i%
```

Quando QBasic incontra l'enunciato NEXT, incrementa il contatore del numero che è stato specificato dopo STEP.

Il numero della clausola STEP non deve essere necessariamente positivo. Se si indica un numero negativo dopo STEP, QBasic fa diminuire il valore iniziale ogni volta che incontra l'enunciato NEXT.

Il valore STEP influisce anche sui valori assegnati a *inizio* e *fine*:

- Se il valore STEP è positivo, il valore di *inizio* deve essere minore o uguale al valore di *fine*.
- Se il valore STEP è negativo, il valore di *inizio* deve essere maggiore o uguale al valore di *fine*.

Riflettete un momento: se si usa un valore negativo per STEP e *fine* è maggiore di *inizio*, QBasic dà per scontato che il lavoro è concluso e dunque salta il blocco degli enunciati senza eseguirlo e continua a eseguire il resto del programma.

Esercizio: Uso di STEP

1. Scrivete il programma FOR-6.BAS (figura 6-6).

```
' FOR-6.BAS
' Questo programma illustra l'uso di STEP.

CLS

FOR i% = 15 TO 1 STEP -1
     COLOR i%
     PRINT "Caduta!"
     SOUND (50 * i%), 1
NEXT i%
```

Figura 6-6. *Uso di STEP in un enunciato FOR.*

2. Eseguite il programma e QBasic mostrerà la parola *Caduta!* 15 volte, ogni volta in un colore diverso e con un suono di accompagnamento.

Nidificare i loop di tipo FOR

Nidificare significa inserire un loop all'interno di un altro loop. Quello che segue è un esempio di un loop di tipo FOR nidificato:

```
FOR i% = 1 TO 5
     FOR j% = 1 TO 5
          PRINT "Vedrai questo messaggio 25 volte."
     NEXT j%
NEXT i%
```


In questo esempio il rientro aiuta davvero a capire che cosa sta succedendo. In un loop di tipo FOR nidificato, il loop di tipo FOR interno è un enunciato per il loop esterno; cioè, il loop di tipo FOR esterno esegue il loop interno per il numero di volte specificato. Nell'esempio, il loop esterno esegue il loop interno per un totale di 5 volte, ma poiché il loop interno esegue a sua volta gli enunciati per 5 volte, QBasic finisce con l'eseguire l'enunciato PRINT 25 volte in totale.

Esercizio: Uso di loop di tipo FOR nidificati

1. Scrivete il programma FOR-7.BAS (figura 6-7).

```
' FOR-7.BAS
' Questo programma mostra l'uso dei loop di tipo FOR nidificati.

CLS

PRINT "Il loop interno stamperà i numeri colorati:"
PRINT

FOR i% = 1 TO 5
     COLOR 7    ' Imposta il colore del loop esterno in bianco
     PRINT ' Stampa una riga vuota
     PRINT "Numero dei cicli del loop esterno"; i%
     FOR j% = 1 TO 10
          COLOR j%
          PRINT j%
          SOUND (j% * 100), 1
     NEXT j%
NEXT i%
```

Figura 6-7. *Uso di loop di tipo FOR nidificati.*

2. Eseguite il programma per ottenere questo risultato:

```
Il loop interno stamperà i numeri colorati:

Numero dei cicli del loop esterno 1
1 2 3 4 5 6 7 8 9 10
Numero dei cicli del loop esterno 2
1 2 3 4 5 6 7 8 9 10
```

```
Numero dei cicli del loop esterno 3
1 2 3 4 5 6 7 8 9 10
Numero dei cicli del loop esterno 4
1 2 3 4 5 6 7 8 9 10
Numero dei cicli del loop esterno 5
1 2 3 4 5 6 7 8 9 10
```

L'enunciato SOUND crea un suono dopo che ogni numero del loop interno è stato stampato. Premete un tasto qualsiasi per tornare alla Finestra di digitazione, cambiate dei valori e cercate di prevedere che cosa farà il programma che avete cambiato prima di eseguirlo; poi eseguite il programma per vedere se avevate indovinato.

L'AFFERMAZIONE WHILE

Per progettare un loop che si ripete finché non si verifica una specifica condizione, si usa l'enunciato WHILE, che termina sempre con l'enunciato WEND, come si vede nella seguente sintassi:

```
WHILE condizione
     enunciati che devono essere eseguiti ripetutamente
WEND
```

condizione può essere una qualunque espressione condizionale (come *anno% < = 1959*, o *temperatura% < 0*).

Quando incontra l'enunciato WHILE, QBasic valuta *condizione*:

- *condizione* è vera, QBasic esegue il blocco delle enunciati tra WHILE e WEND e poi controlla nuovamente *condizione.*
- *condizione* è falsa, QBasic salta subito all'enunciato WEND e va avanti eseguendo il resto del programma.

Per usare un loop WHILE con successo, bisogna provvedere a modificare la condizione che segue l'enunciato WHILE *all'interno* del loop.

NOTA: È necessario assicurarsi che la condizione alla fine diventi falsa; in caso contrario il loop non si fermerà mai e il resto del programma non verrà eseguito (un loop che non si arresta si chiama loop infinito. Per fermare un loop infinito basta tenere premuto il tasto Ctrl e premere Pausa.)

Esercizio: Uso del loop di tipo WHILE

Nel programma che segue, ogni attività ha luogo all'interno del loop e poiché QBasic aumenta il valore di *numUtente%* ogni volta che esegue il loop, *numUtente%* diventa alla fine maggiore di 12. Tutto ciò crea una falsa condizione e fa sì che QBasic cessi di eseguire il loop.

1. Scrivete il programma WHILE.BAS (figura 6-8).

```
' WHILE.BAS
' Questo programma illustra l'uso del loop di tipo WHILE.

CLS

INPUT "Scrivi un numero da 1 a 10: ", numUtente%
PRINT

WHILE numUtente% <= 12
      COLOR numUtente%
      PRINT "Il valore di numUtente% è"; numUtente%
      SOUND ((numUtente% * 30) + 300), .5
      numUtente% = numUtente% + 1
      PRINT
WEND
```

Figura 6-8. *Uso di un loop di tipo WHILE.*

2. Eseguite il programma; lo schermo dell'output dovrebbe apparire così:

```
Scrivi un numero da 1 a 10: 9

Il valore di numUtente% è 9

Il valore di numUtente% è 10

Il valore di numUtente% è 11

Il valore di numUtente% è 12
```

Nidificazione di loop di tipo WHILE

Si è già visto che è possibile nidificare gli enunciati FOR; la stessa cosa si può fare con gli enunciati WHILE. Ogni enunciato WHILE ha una sua *condizione* e si chiude con l'enunciato WEND; non è necessario che le due condizioni siano collegate.

La sintassi per un loop di tipo WHILE nidificato è la seguente:

```
WHILE condizione
      WHILE condizione
            enunciato per il loop di tipo WHILE interno
      WEND
WEND
```

Il loop di tipo WHILE interno svolge funzione di enunciato per quello esterno.

NOTA: Per evitare loop infiniti, si ricordi che sia la condizione interna che quella esterna devono alla fine risultare false.

L'ENUNCIATO DO

L'enunciato DO viene impiegato con l'enunciato LOOP per sviluppare due tipi di loop:

- Un loop che si ripete *finché* una certa condizione resta vera (DO WHILE).
- Un loop che si ripete *sino a che* una certa condizione non risulta vera (DO UNTIL).

DO WHILE: il loop che si ripete finché una condizione resta vera

Per creare un loop che si ripete finché una condizione resta vera, si deve usare un loop di tipo DO con la seguente sintassi:

```
DO WHILE condizione
      blocco delle enunciati che devono essere eseguiti
LOOP
```


Un loop DO di questo tipo funziona allo stesso modo dell'enunciato WHILE. QBasic esamina la *condizione*:

- Se *condizione* è vera, QBasic esegue gli enunciati e continua a farlo finché la *condizione* rimane vera.
- Se *condizione* è falsa, QBasic salta l'enunciato LOOP ed esegue il resto del programma.

Il rapporto condizione-enunciato

Come si è visto nel capitolo 2, QBasic esegue le istruzioni nell'ordine in cui appaiono nei programmi. L'ordine è particolarmente importante nel caso dell'enunciato DO; in tale enunciato, infatti, si pone una condizione in una posizione o in un'altra, a seconda del risultato che si vuole ottenere dal programma:

- Se si vuole che gli enunciati eseguano i loro compiti soltanto sulla base del valore di *condizione*, si deve mettere *condizione* sulla riga dell'enunciato DO:

```
DO {WHILE:UNTIL} condizione
  blocco degli enunciati che devono essere eseguiti
LOOP
```

  Con questa impostazione gli enunciati interni vengono eseguiti solo se *condizione* è vera.

- Se si vuole che gli enunciati eseguano almeno una volta i loro compiti, qualunque sia il valore di *condizione*, si deve mettere *condizione* sulla riga dell'enunciato LOOP:

```
DO
  blocco degli enunciati che devono essere eseguiti
LOOP {WHILE:UNTIL} condizione
```

  Quando *condizione* viene presa in considerazione, QBasic ha già eseguito una volta gli enunciati.

Esercizio: Uso di un loop di tipo DO WHILE con la condizione all'inizio

1. Scrivete il programma DO-1.BAS (figura 6-9).

```
' DO-1.BAS
' Questo programma mostra l'uso di un loop di tipo DO WHILE.
' Si noti che la condizione è all'inizio del loop.

CLS

INPUT "Scrivi un numero da 10 a 15: ", numUtente%
PRINT

DO WHILE numUtente% >= 10
      COLOR numUtente%
      PRINT "Il valore di numUtente% è"; numUtente%
      numUtente% = numUtente% - 1 ' Diminuisce numUtente% di 1
LOOP
```

Figura 6-9. *Uso di un loop di tipo DO WHILE.*

2. Eseguite il programma; lo schermo dell'output dovrebbe apparire così:

```
Scrivi un numero da 10 a 15: 13

Il valore di numUtente% è 13
Il valore di numUtente% è 12
Il valore di numUtente% è 11
Il valore di numUtente% è 10
```

3. Eseguite nuovamente il programma e scrivete un valore uguale o minore di 9. Il risultato sarà il seguente:

```
Scrivi un numero da 10 a 15: 6
```

QBasic si è accorto che la condizione era falsa e ha saltato gli enunciati all'interno del loop.

Esercizio: Uso di un loop di tipo DO WHILE con la condizione alla fine

Il programma DO-2.BAS è identico a DO-1.BAS ma la condizione è stata spostata alla fine del loop.

1. Modificate il programma DO-1.BAS in DO-2.BAS (figura 6-10).

2. Eseguite il programma e osservate il risultato:


```
' DO-2.BAS
' Questo programma mostra l'uso di un loop di tipo DO WHILE.
' Si noti che la condizione è alla fine del loop.

CLS

INPUT "Scrivi un numero da 10 a 15: ", numUtente%
PRINT

DO
      COLOR numUtente%
      PRINT "Il valore di numUtente% è"; numUtente%
      numUtente% = numUtente% - 1 ' Diminuisce numUtente% di 1
LOOP WHILE numUtente% >= 10
```

Figura 6-10. *Un loop di tipo DO WHILE con la condizione alla fine.*

```
Scrivi un numero da 10 a 15: 13

Il valore di numUtente% è 13
Il valore di numUtente% è 12
Il valore di numUtente% è 11
Il valore di numUtente% è 10
```

Sembrerebbe in tutto simile a DO-1.BAS, ma non è così. Vediamo infatti come continua l'esercizio:

3. Eseguite nuovamente il programma ma questa volta scrivete un valore minore o uguale a 9. Lo schermo output apparirà in questo modo:

```
Scrivi un numero da 10 a 15: 6

Il valore di numUtente% è 6
```

Qui si vede la differenza: anche se il valore di *numUtente%* ha fatto sì che la condizione nell'enunciato LOOP fosse falsa, QBasic non ha incontrato *condizione* se non *dopo* aver eseguito una volta il corpo del loop di tipo DO.

DO UNTIL: il loop che si ripete sino a che una condizione non risulta vera

Per creare un loop che esegue i suoi compiti sino a che una certa condizione non si verifica, bisogna usare un loop di tipo DO con la seguente sintassi:

```
DO UNTIL condizione
     blocco degli enunciati che devono essere eseguiti
LOOP
```

QBasic esamina la condizione:

- Se *condizione* è vera, QBasic salta l'enunciato LOOP per eseguire il resto del programma.
- Se *condizione* è falsa, QBasic esegue gli enunciati e continua a farlo sino a che *condizione* non risulta vera.

Esercizio: Uso di un loop di tipo DO UNTIL con la condizione all'inizio

1. Scrivete il programma DO-3.BAS (figura 6-11).

```
' DO-3.BAS
' Questo programma mostra l'uso di un loop di tipo DO UNTIL.
' Si noti che la condizione è all'inizio del loop.

CLS

INPUT "Scrivi un numero da 1 a 10: ", numUtente%
PRINT

DO UNTIL numUtente% > 10
      COLOR numUtente%
      PRINT "Il valore di numUtente% è"; numUtente%
      numUtente% = numUtente% + 1  ' Incrementa numUtente% di 1
LOOP
```

Figura 6-11. *Uso di un loop di tipo DO UNTIL.*

2. Eseguite il programma; lo schermo output dovrebbe apparire così:


```
Scrivi un numero da 1 a 10: 7

Il valore di numUtente% è 7
Il valore di numUtente% è 8
Il valore di numUtente% è 9
Il valore di numUtente% è 10
```

Dal momento che la condizione *numUtente% > 10* è falsa quando QBasic incontra il loop per la prima volta, questo viene eseguito sino a che la condizione risulta vera.

3. Eseguite nuovamente il programma, questa volta scrivendo un valore uguale o maggiore a 11. Lo schermo dell'output apparirà in questo modo:

```
Scrivi un numero da 1 a 10: 13
```

Questa volta, poiché la condizione è vera quando QBasic incontra il loop, questo viene saltato e QBasic prosegue con l'esecuzione del programma.

Esercizio: Uso di un loop di tipo DO UNTIL con la condizione alla fine

1. Modificate il programma DO-3.BAS in DO-4.BAS (figura 6-12).

```
' DO-4.BAS
' Questo programma illustra l'uso di un loop di tipo DO UNTIL.
' Si noti che la condizione è alla fine del loop.

CLS

INPUT "Scrivi un numero da 1 a 10: ", numUtente%
PRINT

DO
      COLOR numUtente%
      PRINT "Il valore di numUtente% è"; numUtente%
      numUtente% = numUtente + 1   ' Incrementa numUtente% di 1
LOOP UNTIL numUtente% > 10
```

Figura 6-12. *Un loop di tipo DO UNTIL con la condizione alla fine.*

2. Eseguite il programma e osservate il risultato:

```
Scrivi un numero da 1 a 10: 7

Il valore di numUtente% è 7
Il valore di numUtente% è 8
Il valore di numUtente% è 9
Il valore di numUtente% è 10
```

Il risultato di questo programma è lo stesso di quello precedente.

3. Eseguite nuovamente il programma scrivendo un valore uguale o maggiore di 11. Ecco l'output del programma:

```
Scrivi un numero da 1 a 10: 13

Il valore di numUtente% è 13
```

Questa volta, poiché la condizione era alla fine del loop, QBasic ha eseguito una volta i contenuti del loop non sapendo quale fosse la condizione sino a quando non l'ha incontrata alla fine del loop.

Nidificare loop di tipo DO

Si possono nidificare i loop di tipo DO allo stesso modo dei loop di tipo FOR e di tipo WHILE.

Dal punto di vista dell'impostazione i loop non sono assolutamente collegati; in altre parole, se si hanno due loop di tipo DO nidificati e si mette la condizione all'inizio del loop esterno, non è necessario metterla anche all'inizio del loop interno. Naturalmente, come per un singolo loop di tipo DO, bisogna assicurarsi che sia il loop interno che il loop nidificato abbiano termine.

NIDIFICARE TIPI DIVERSI DI LOOP

Finora si è parlato di nidificare loop dello stesso tipo, un loop di tipo FOR all'interno di un altro loop di tipo FOR, un loop di tipo WHILE all'interno di un altro loop di tipo WHILE e così via.

Si possono anche mescolare i tipi di loop, specialmente se ciò contribuisce a rendere il programma più efficiente. Il programma che segue inserisce un loop di tipo FOR all'interno di un loop di tipo DO. Il loop di tipo DO permette di eseguire il loop di tipo FOR quante volte si desidera senza dover ricominciare il programma ogni volta.

Esercizio: Nidificare tipi diversi di loop

1. Scrivete il programma NIDIFICA.BAS (figura 6-13).

```
' NIDIFICA.BAS
' Questo programma mostra la nidificazione di tipi diversi di
loop.

CLS

DO
      PRINT "Quanti toni vuoi sentire?"
      INPUT "(Digita 0 per finire) ", numSuoni%
      PRINT

      FOR i% = 1 TO numSuoni%
            SOUND (i% * 100), 1
      NEXT i%

LOOP UNTIL numSuoni% = 0
```

Figura 6-13. *Nidificare tipi diversi di loop.*

2. Eseguite il programma; l'output del programma dovrebbe essere il seguente:

```
Quanti toni vuoi sentire?
(Digita 0 per finire)  5

Quanti suoni vuoi sentire?
(Digita 0 per finire)  20

Quanti suoni vuoi sentire?
(Digita 0 per finire)  0
```

USI PRATICI DEI LOOP DI QBASIC

Dopo aver cominciato a conoscere i loop, passiamo a vedere i loro usi pratici. Gli esempi che seguono presentano alcune pratiche e divertenti applicazioni dei loop.

Uso di un loop per raccogliere informazioni

Supponiamo che si voglia automatizzare la preparazione di un budget mensile. La filosofia che sta dietro alla preparazione di un budget di questo tipo è chiara: si sottraggono mensilmente le spese dalle entrate.

Usando gli strumenti di cui si è parlato finora, si può scrivere un programma in QBasic che può essere d'aiuto nello svolgere questo compito. Si possono usare alcuni enunciati INPUT: uno per la somma iniziale e uno per ogni spesa; fatti alcuni calcoli, QBasic stamperà quanti soldi sono rimasti. La figura 6-14 mostra un esempio di questo tipo di programma.

Quale loop scegliere?

Cercate di tenere a mente le seguenti indicazioni quando dovete decidere quale tipo di loop usare in un programma:

- Si usa un loop di tipo FOR quando si vuole eseguire un blocco di enunciati uno specifico numero di volte.
- Si usa un loop di tipo WHILE o DO per eseguire degli enunciati che si basano sul valore di una condizione.

Un loop di tipo DO può fare tutto ciò che può fare un loop di tipo WHILE. Cercate di abituarvi a usare, quando possibile, il loop di tipo DO, che è più versatile (QBasic si serve ancora del loop di tipo WHILE per permettere di eseguire programmi che sono stati scritti per le prime versioni del BASIC).

Questo programma funziona abbastanza bene ma presenta dei limiti:

- Poiché il numero delle spese può variare di mese in mese, si dovrà cambiare il programma ogni volta che lo si esegue.
- Ogni spesa che si scrive deve essere subito sottratta dal totale; ciò non comporta grossi problemi, ma dà luogo a una serie di calcoli ripetitivi. Basti solo immaginare quanto lungo sarebbe il programma se ci fossero 20 o più spese ogni mese.

I programmi che presentano simili ripetizioni sono i candidati ideali per un loop. Il programma BUDGET.BAS della figura 6-15 compie lo stesso lavoro del programma presentato nella figura 6-14 ma offre alcuni vantaggi:


```
' Questo programma aiuta a calcolare quanti soldi
' rimangono dopo aver pagato le bollette mensili.

CLS

COLOR 2      ' Primo piano verde
PRINT "Calcolatrice di budget"
COLOR7       ' Ripristina il colore bianco predefinito
PRINT
INPUT "Scrivi il totale delle entrate di questo mese: $",
totale!
PRINT

INPUT "Scrivi spesa n.1: L", spesal!
totale! = totale! - spesal!
INPUT "Scrivi spesa n.2: L", spesa2!
totale! = totale! - spesa2!
INPUT "Scrivi spesa n.3: L", spesa3!
totale! = totale! - spesa3!
INPUT "Scrivi spesa n.4: L", spesa 4!
totale! = totale! - spesa4!

PRINT
PRINT "Ti rimangono L"; totale!
```

Figura 6-14. *Un esempio di programma di budget.*

- È un programma più breve.
- Prevede un numero variabile di spese in base al numero totale di bollette che pensate di inserire.
- Usa un loop sia per chiedere la somma delle bollette sia per sottrarre la somma dal totale.

Uso di loop con numeri in ordine casuale

Dadi, roulette, bingo, lotteria: tutti questi giochi fanno uso di *numeri che ricorrono secondo un ordine casuale*, un ordine che non si può predire. In questo paragrafo vedremo gli strumenti di cui dispone QBasic per creare dei numeri casuali: RND, RANDOMIZE TIMER e INT.

```
' BUDGET.BAS
' Questo programma aiuta a calcolare quanti soldi
' rimangono dopo aver pagato le bollette mensili.

CLS

COLOR 2     ' primo piano verde
PRINT "Calcolatrice di budget"
COLOR 7     ' ripristina il colore di fondo bianco predefinito
PRINT
INPUT "Scrivi il totale delle entrate di questo mese: L",
totale!
PRINT
INPUT "Quante bollette avrai questo mese? ", bollette%

FOR i% = 1 TO bollette%
        PRINT "Scrivi la spesa n."; i%;
        INPUT ": L", questaSpesa!
        totale! = totale! - questaSpesa!
NEXT i%

PRINT
PRINT "Ti rimangono L"; totale!
```

Figura 6-15. *Un programma di budget più efficiente.*

Generazione di numeri in ordine casuale

La funzione RND indica a QBasic di restituire un numero scelta a caso; il numero sarà un numero in virgola mobile a precisione singola tra 0 e 1. RND è una funzione, e perciò deve essere usata all'interno di un enunciato di QBasic. La sintassi per la forma più semplice della funzione RND è questa:

```
RND
```

RND restituisce la stessa serie di numeri ogni volta che si esegue il programma. Per creare ogni volta una serie di numeri *diversi*, si usa RANDOMIZE TIMER come uno dei primi enunciati del programma. La sintassi per l'enunciato RANDOMIZE TIMER è semplice come quella della funzione RND:

```
RANDOMIZE TIMER
```


Esercizio: Come generare dei numeri in ordine casuale

Il programma RND-1.BAS (figura 6-16) usa un loop di tipo FOR per creare sei numeri a caso (per creare più numeri a caso, cambiate semplicemente il 6 dell'enunciato FOR in un valore più alto).

1. Scrivete il programma RND-1.BAS.

```
' RND-1.BAS
' Questo programma genera numeri a caso tra 0 e 1.

CLS

RANDOMIZE TIMER

FOR i% = 1 TO 6
      PRINT RND
NEXT i%
```

Figura 6-16. *Scelta causale di numeri.*

2. Eseguite il programma e otterrete un risultato simile a questo:

```
.2494013
.3081127
3.670245E-02
.574261
.4791026
.1495265
```

I numeri che otterrete saranno senza dubbio diversi da questi, proprio perché sono dei numeri scelti a caso.

Personalizzazione dei risultati di RND

Poiché è probabile che i numeri tra 0 e 1 non soddisfino le esigenze di tutti, si possono usare gli operatori matematici di QBasic per cambiare la grandezza dei risultati di RND.

Esercizio: Come generare dei numeri di dimensioni maggiori in ordine casuale

Il programma RND-2.BAS (figura 6-17) è identico a RND-1.BAS, ma moltiplica il risultato di RND per 100, creando dei valori che vanno da 0 a 100.

1. Modificate il programma RND-1.BAS in RND-2.BAS.

```
' RND-2.BAS
' Questo programma genera numeri a caso tra 0 e 100.

CLS

RANDOMIZE TIMER

FOR i% = 1 TO 6
     PRINT RND * 100
NEXT i%
```

Figura 6-17. *Generare numeri di dimensioni maggiori in ordine casuale.*

2. Eseguite il programma per avere un risultato simile a questo:

```
85.67621
42.35302
87.795
48.80183
42.53306
24.83424
```

Come generare dei numeri interi in ordine casuale

Se si desidera generare numeri interi in ordine casuale, bisogna usare la funzione INT insieme alla funzione RND. INT scarta la parte decimale di un numero, mantenendo soltanto la parte intera; il formato è il seguente:

```
INT(numero)
```

numero è il numero decimale che si vuole arrotondare. Come tutte le funzioni che restituiscono numeri interi, il risultato intero di INT deve essere assegnato o a un enunciato che accetta valori interi o a una variabile intera.

Numeri molto piccoli o molto grandi in QBasic

I valori prodotti dalla funzione RND sono sempre maggiori di 0 e inferiori a 1. Talvolta sono valori molto piccoli.

Quando QBasic cerca di stampare un valore che è troppo piccolo o troppo grande per essere visualizzato senza usare molte cifre, usa la sua numerazione esponenziale. Più semplicemente, quando si incontra un numero con una E o una D, significa che il punto decimale è stato spostato dalla sua posizione consueta. Il numero dopo la E o la D indica in quale direzione e per quanti posti il punto decimale è stato spostato. Se quel numero è positivo, il punto decimale si trova a destra della posizione corrente; se il numero è negativo, si trova a sinistra.

Per esempio, nell'esercizio RND-1.BAS si trova il numero 3.670245E-02. E-02 indica che il punto decimale si trova spostato a sinistra di due posti. In altre parole, nella notazione consueta il numero sarebbe 0.03670245.

Ci si chiederà allora perché QBasic non visualizzi semplicemente i numeri così come siamo abituati a vederli. Spesso la schermo è appena sufficiente. Si prenda per esempio il numero 4.136111E28: se venisse stampato per intero, apparirebbe in questo modo:

```
41361110000000000000000000000
```

Per coloro che sono pratici di notazione scientifica, ecco come apparirebbe lo stesso numero:

4.136111×10^{28}

Gli esercizi che seguono mostrano come usare la funzione INT insieme alla funzione RND per generare numeri interi in ordine casuale.

Esercizio: Il gioco "indovina un numero"

I numeri generati secondo un ordine casuale sono ideali per i giochi in cui bisogna indovinare un numero. Questo programma usa un loop di tipo DO:

1. Scrivete il programma INDOVINA.BAS (figura 6-18).
2. Eseguite il programma; l'output del programma dovrebbe apparire così:

```
Il gioco Indovina Un Numero
```

```
Sto pensando a un numero da 1 a 100.
Sei capace di indovinarlo?

Che numero è? 50
Prova un numero più grande!

Che numero è? 99
Prova un numero più piccolo!

Che numero è? 65
Prova un numero più grande!

Che numero è? 71
Esatto!!!

Hai indovinato il numero con 4 tentativi!
```

Esercizio: Scrivere una simulazione al computer

Poniamo il caso si voglia scrivere un programma che calcoli quante volte esce il numero 7 tirando 100 volte 2 dadi. La funzione RND si rivela ancora una volta utile, come si può vedere nel seguente programma DADI.BAS (figura 6-19).

1. Scrivete il programma DADI.BAS.
2. Eseguite il programma; l'output del programma dovrebbe essere:

```
Programma simulazione-dadi

Quante volte devo tirare i dadi? 50

Sto lavorando...

Tirando 50 volte, è uscito il numero 7
per 10 volte.
```

SOMMARIO

I loop permettono di creare programmi potenti che ripetono un compito uno specifico numero di volte oppure fino a quando incontrano una condizione. In questo capitolo abbiamo visto i loop di tipo FOR, WHILE e DO. Con le


```
' INDOVINA.BAS
' Questo ™ il programma per il gioco "Indovina un numero". Il programma
' genera un numero a caso e chiede all'utente di indovinare qual'è.
' Dopo che l'utente ha indovinato il numero, il programma visualizza
' il numero di tentativi fatti.

CLS

PRINT "Il gioco Indovina un numero"
PRINT
PRINT "Sto pensando a un numero da 1 a 100."
PRINT "Sei capace di indovinarlo?"
PRINT

RANDOMIZE TIMER

numAcaso% = INT(RND * 100) + 1  ' Genera un numero a caso
numTentativi% = 0  ' Inizia da capo

DO
        INPUT "Che numero è? ", indovina%
        IF indovina% = numAcaso% THEN PRINT "Esatto!!!"
        IF indovina% < numAcaso% THEN PRINT "Prova un numero più grande!"
        IF indovina% > numAcaso% THEN PRINT "Prova un numero più piccolo!"

        numTentativi% = numTentativi% + 1     ' Registra i tentativi fatti
        PRINT ' Stampa una riga vuota

LOOP UNTIL indovina% = numAcaso%

PRINT "Hai indovinato il numero con"; numTentativi%; "tentativi!"
```

Figura 6-18. *Un gioco in cui bisogna indovinare un numero.*

```
' DADI.BAS
' Questo programma chiede all'utente di scrivere quante
' volte QBasic deve "tirare" due dadi, e poi calcola
' quante volte esce il numero 7.

CLS

RANDOMIZE TIMER

COLOR 4     ' Colore rosso per il titolo
PRINT "Programma simulazione-dadi"
COLOR 7     ' Ripristina il colore bianco predefinito
PRINT
numSette% = 0     ' Inizia a calcolare da 0

INPUT "Quante volte devo tirare il dado? ", volte%
PRINT
PRINT "Sto lavorando..."      ' Si assicura che l'utente
                              ' sappia che tutto va bene

FOR i% = 1 TO volte%
      dado1% = INT(RND * 6) + 1     '"Tira" il primo dado
      dado2% = INT(RND * 6) + 1     '"Tira" il secondo dado
      IF dado1% + dado2% = 7 THEN numSette% = numSette% + 1
NEXT i%

PRINT
PRINT "Tirando"; volte%; "volte, è uscito il numero 7"
PRINT "per"; numSette%; "volte."
```

Figura 6-19. *Un programma che simula il lancio di due dadi.*

enunciati COLOR e SOUND, la funzione RND e con quanto si è appreso nei capitoli precedenti si è ora in grado di intraprendere dei progetti di programmazione di sicuro effetto, come si vedrà nei prossimi capitoli.

DOMANDE ED ESERCIZI

1. Qual è lo scopo di una variabile di calcolo in un loop di tipo FOR?

2. Quali valori possono essere assegnati agli elementi di *inizio* e *fine* in un loop di tipo FOR?

3. Quale numero bisogna specificare come argomento dell'enunciato COLOR se si vuole mettere il colore rosso nello sfondo?

4. Che cosa fa l'enunciato DO?

5. Che cos'è un loop nidificato?

6. Che effetto si ottiene mettendo la condizione in un loop di tipo DO all'inizio del loop? Che effetto si ottiene mettendola alla fine?

7. In quali circostanze si usa un loop di tipo FOR? E un loop di tipo WHILE o di tipo DO?

8. Scrivete un programma che tenga nota dei soldi spesi in benzina in una settimana. Usate un loop di tipo FOR per ordinare le banconote e le monete spese e usate come totale una variabile in virgola mobile a precisione singola.

9. Scrivete un programma che richieda all'utente una frequenza valida e i valori di durata per rieseguirli con l'enunciato SOUND. Usate un loop di tipo DO UNTIL per riunire le informazioni sino a che l'utente digita -999 come frequenza.

10. Scrivete un programma che simuli il lancio di un dado per 10 volte. Stampate il valore del dado dopo ogni tiro e visualizzate il messaggio *Bel tiro!* in verde se esce 6.

CAPITOLO 7

Creare sottoprogrammi e funzioni proprie

Fino a questo punto abbiamo digitato programmi relativamente brevi (nessuno più lungo di 30 righe) ma ora, dopo aver appreso i fondamenti di QBasic, possiamo affrontare programmi più lunghi. In questo capitolo verranno spiegate alcune tecniche che permettono di scrivere programmi di maggiori dimensioni con un minimo sforzo e senza grosse perdite di tempo.

Analizzeremo due strutture di programma: i *sottoprogrammi* e le *funzioni* (vedremo in particolare come queste differiscano dalle funzioni interne usate sino ad ora): queste due strutture vengono usate per separare il programma in due unità di facile impiego dette *procedure*. Ci occuperemo inoltre della dichiarazione di sottoprogrammi e funzioni e del loro uso nei programmi; questo ci permetterà anche di spiegare come l'ambiente QBasic si adatti alla programmazione modulare. Infine ci serviremo di tutti questi strumenti per dar vita a programmi efficienti e ben strutturati.

PERCHÉ SI USANO PROCEDURE?

Supponiamo di voler scrivere un programma che stampa le parole della canzone popolare "John Brown". Basandoci solo sulle conoscenze acquisite fino a questo momento dovremmo scrivere la canzone usando un gran numero di enunciati PRINT, come nel programma CANZONE.BAS (figura 7-1).

CANZONE.BAS è abbastanza elementare e di facile comprensione: stampa ogni verso della canzone fermandosi quando lo schermo è pieno per consentire una più comoda lettura e poi passa alla schermata successiva.

Si noti che il programma CANZONE.BAS contiene un ritornello che viene ripetuto molte volte senza alcuna modifica; è evidente che un testo ripetitivo non solo richiede tempo per venir digitato ma rende anche più confuso il listato e più difficile il lavoro. Esiste un modo più semplice per codificare un programma che contiene parti ripetitive?

Il vantaggio di usare una procedura

La risposta alla domanda precedente è sì! QBasic fornisce una struttura di programmazione detta *procedura* che permette di scrivere un blocco di enunciati, assegnargli un nome e poi richiamarlo quando si vuole che il programma lo esegua. Le procedure presentano questi vantaggi:

- **Le procedure eliminano la ripetizione di righe**. Basta infatti definire una procedura una volta per poi eseguirla tutte le volte che si vuole.


```
' CANZONE.BAS
' Questo programma mostra le parole della canzone "John Brown"
CLS
PRINT "------------------------John Brown------------------------"
PRINT
PRINT "John Brown giace nella tomba là nel pian"  'Prima strofa
PRINT "dopo una lunga lotta contro l'oppressor."
PRINT "John Brown giace nella tomba là nel pian"
PRINT "ma l'anima vive ancor.'
PRINT
PRINT "Glory, glory alleluia"  Ritornello
PRINT "Glory, glory alleluia"
PRINT "Glory, glory alleluia"
PRINT "ma l'anima vive ancor
PRINT
PRINT "Con 19 suoi compagni di valor" 'Seconda strofa
PRINT "dall'Est all'Ovest la Virginia conquistò"
PRINT "con 19 suoi compagni di valor"
PRINT "ma l'anima vive ancor..."
PRINT
PRINT "Glory, glory alleluia"  'Ritornello
PRINT "Glory, glory alleluia"
PRINT "Glory, glory alleluia"
PRINT "ma l'anima vive ancor
PRINT
INPUT "Premere Invio per continuare...", risposta$ 'Pausa
PRINT
PRINT "Poi lo hanno ucciso come fosse un traditor"Terza strofa
PRINT "ma traditore fu colui che lo impiccò;"
PRINT "poi lo hanno ucciso come fosse un traditor"
PRINT "ma l'anima vive ancor..."
PRINT
PRINT "Glory, glory alleluia"  'Ritornello
PRINT "Glory, glory alleluia"
PRINT "Glory, glory alleluia"
PRINT "ma l'anima vive ancor
PRINT
PRINT "John Brown è morto ma lo schiavo è in libertà"   'Quarta strofa
PRINT "tutti fratelli bianchi e neri siamo già."
PRINT "John Brown è morto ma lo schiavo è in libertà"
PRINT "ma l'anima vive ancor..."
PRINT
PRINT "Glory, glory alleluia"  'Ritornello
```

Figura 7-1. *Un programma che stampa le parole della canzone "John Brown" tramite una serie di enunciati PRINT (continua).*

```
PRINT "Glory, glory alleluia"
PRINT "Glory, glory alleluia"
PRINT "ma l'anima vive ancor"
PRINT
```

Figura 7-1. *Un programma che stampa le parole della canzone "John Brown" tramite una serie di enunciati PRINT.*

- **Le procedure rendono il programma più semplice da leggere.** Un programma diviso in parti più piccole è di più facile comprensione.
- **Le procedure semplificano lo sviluppo del programma.** Programmi suddivisi in unità logiche sono più facili da progettare, scrivere e "ripulire" (debug); inoltre programmatori diversi possono usare le stesse procedure.
- **Le procedure possono essere usate in programmi differenti.** Si possono creare procedure generali da usare molte volte in contesti diversi.
- **Le procedure estendono il linguaggio QBasic.** Infatti possono svolgere compiti che non potrebbero essere realizzati direttamente con gli enunciati e le funzioni interne di QBasic.

In questo capitolo vedremo come produrre e impiegare due procedure di QBasic: sottoprogrammi e funzioni. I primi consentono di suddividere il programma in unità più piccole che possono essere richiamate più volte; le seconde forniscono dei valori utilizzabili dal programma.

CREAZIONE DI SOTTOPROGRAMMI

I sottoprogrammi rendono il codice più facile da leggere e riducono le ripetizioni. Un sottoprogramma è un blocco di codice racchiuso tra gli enunciati SUB e END SUB e può essere richiamato all'interno del programma quante volte si vuole. Quando un sottoprogramma finisce, il programma riparte dall'enunciato che segue il codice di chiamata del sottoprogramma. QBasic permette di creare e conservare i sottoprogrammi separatamente dal programma principale.

Sintassi di un sottoprogramma

La sintassi di un sottoprogramma può definirsi in questo modo:

```
SUB NomeSottoprogramma (parametroLista)
    dichiarazioni di variabili e costanti locali
    Enunciati del sottoprogramma
END SUB
```

- *SUB* è l'enunciato di QBasic che indica l'inizio della definizione del sottoprogramma.
- *NomeSottoprogramma* è il nome del sottoprogramma. Può avere una lunghezza massima di 40 caratteri ed è il nome che il programma usa per "chiamare" il sottoprogramma. Non può corrispondere a una parola chiave di QBasic né al nome di una variabile o di una procedura del programma.
- *(parametroLista)* è un elenco facoltativo di variabili (si veda, a questo proposito, il paragrafo "Inserimento di argomenti un sottoprogramma") che va racchiusa tra parentesi.
- *dichiarazioni di variabili e costanti locali* è un elenco facoltativo di variabili e costanti dichiarato e usato solo ed esclusivamente all'interno del sottoprogramma. Queste non hanno alcun effetto su variabili o costanti che hanno lo stesso nome ma sono contenute in altre parti del programma.
- *Enunciati del sottoprogramma* è la parte operativa del sottoprogramma dove si possono usare quasi tutti gli enunciati di QBasic.
- *END SUB* è l'enunciato di QBasic che segna la fine della definizione del sottoprogramma.

Per vedere come questi elementi lavorano insieme considerate il seguente sottoprogramma (Ritornello). Ogni volta che viene eseguito, il sottoprogramma stampa una riga vuota, il ritornello della canzone e un'altra riga vuota.

```
SUB Ritornello
' Il sottoprogramma Ritornello stampa il ritornello della canzone
PRINT
PRINT "Glory, glory alleluia"     'Ritornello
PRINT "Glory, glory alleluia"
PRINT "Glory, glory alleluia"
PRINT "ma l'anima vive ancor
PRINT
END SUB
```

Creazione di un sottoprogramma

Nell'ambiente QBasic i menu permettono di creare e di lavorare velocemente e facilmente con i sottoprogrammi. QBasic dispone di due comandi di menu per lavorare con sottoprogrammi:

- Il comando Nuovo SUB (nel menu Modifica)
- Il comando SUBs (nel menu Visualizza)

Il comando Nuovo SUB

Il comando Nuovo SUB viene usato per aggiungere al programma un sottoprogramma. Per creare un sottoprogramma si possono compiere i passi seguenti:

1. Selezionate il comando Nuovo SUB che apre la relativa finestra di dialogo.
2. Inserite il nome del sottoprogramma (se non è già disponibile).
3. Scrivete i parametri e il corpo del programma.

Il comando SUBs

Il comando SUBs permette di esaminare e modificare i programmi. Questo comando apre una finestra di dialogo dalla quale si può selezionare la parte del programma su cui si vuole lavorare: il programma principale, un sottoprogramma o una funzione.

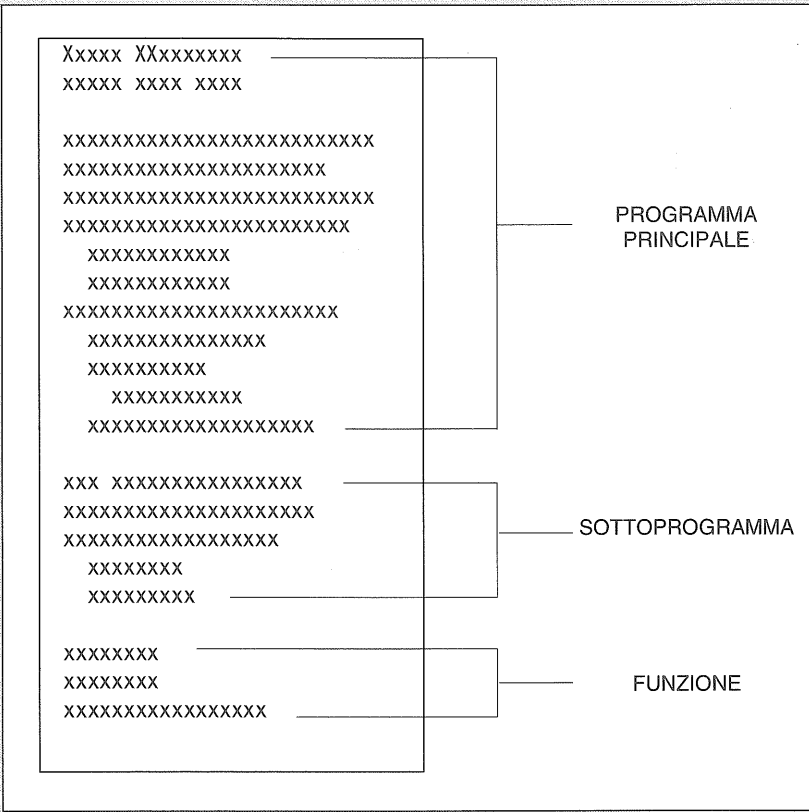
NOTA: Il comando SUBs viene attivato anche premendo F2 (questo è il modo più veloce per aprire la relativa finestra di dialogo).

Richiamare un sottoprogramma

Dopo aver creato e dichiarato un sottoprogramma è possibile "chiamarlo" (eseguirlo). Per chiamare un sottoprogramma basta specificarne il nome (insieme a qualsiasi argomento che il sottoprogramma richiede) all'interno del programma principale o di un'altra procedura.

Il programma CANZONE2.BAS (figura 7-2) mostra come dichiarare e chiamare il sottoprogramma Ritornello. In questa forma CANZONE2.BAS è

Suddividere un programma

Questo paragrafo spiega come organizzare un programma in tre blocchi di codice autonomi: il programma principale, i sottoprogrammi e le funzioni. Un programma può contenere un numero indefinito di sottoprogrammi e funzioni (detti collettivamente *procedure*). Ogni programma comunque deve avere una sezione principale che controlla il flusso generale e chiama i sottoprogrammi e le funzioni.

Per quanto possa richiedere un po' di tempo (almeno all'inizio), non c'è dubbio che imparare a suddividere il programma in queste tre unità è davvero utile quando si scrivono programmi molto lunghi. Si ricordi anche che per quanti sottoprogrammi e funzioni possa contenere, il programma è e resta uno solo.

L'enunciato DECLARE

Uno dei vantaggi di QBasic è che aiuta a programmare correttamente. Per esempio QBasic aggiunge un enunciato DECLARE in cima al programma principale (al momento del salvataggio su disco) in base alle informazioni fornite al momento di creazione del programma. Un enunciato DECLARE indica l'esistenza di un sottoprogramma; per questa ragione a ogni sottoprogramma deve corrispondere un enunciato DECLARE in cima al programma principale.

La sintassi dell'enunciato DECLARE è la seguente:

```
DECLARE SUB NomeSottoprogramma (parametri)
```

NomeSottoprogramma è il nome del sottoprogramma e i *parametri* sono un elenco di variabili accolte dal sottoprogramma.

NOTA: Sebbene QBasic inserisca l'enunciato DECLARE autonomamente, spetta a voi conservare l'elenco dei parametri. Se modificate l'elenco nel sottoprogramma dovrete modificarlo anche all'interno dell'enunciato DECLARE. Si veda il paragrafo "Inserimento di argomenti un sottoprogramma" per maggiori informazioni.

Per convenzione gli enunciati DECLARE vengono inseriti dopo i commenti di introduzione e prima delle dichiarazioni delle variabili e delle costanti; inoltre, QBasic richiede che questi enunciati precedano ogni enunciato eseguibile.

più breve e più facile da leggere. Si noti come il sottoprogramma ritornello venga impostato al di fuori del programma principale (iniziando alla riga 43 tra gli enunciati SUB e END SUB). Il programma principale chiama il sottoprogramma Ritornello cinque volte. Il nome del sottoprogramma inizia qui con una lettera maiuscola: nel corso del libro useremo questa convenzione per distinguere i nomi delle procedure da quelli delle variabili e da quelli di altri enunciati e funzioni di QBasic.

NOTA: La figura 7-2 contiene il listato del programma CANZONE2.BAS come apparirebbe se venisse stampato ma non come appare in realtà sullo schermo di QBasic. Ricordate che in QBasic tutti i sottoprogrammi vengono tenuti separati dal programma principale e sono visibili attraverso il comando SUBs del menu Visualizza Per scrivere questo programma dall'inizio potrete seguire le istruzioni fornite nell'esercizio successivo.

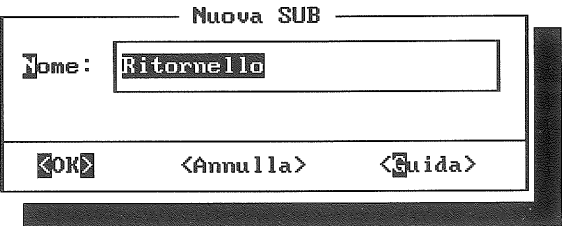
La figura 7-3 mostra la definizione e la chiamata del sottoprogramma Ritornello in dettaglio.

L'enunciato END

Nel corso del libro potrete notare un enunciato END alla fine della sezione del programma principale quando questo contiene sottoprogrammi o funzioni. END è un'istruzione opzionale che indica a QBasic il raggiungimento dell'ultima istruzione e quindi il termine dell'esecuzione. Anche se QBasic non richiede l'enunciato END il suo inserimento può essere utile per indicare visivamente il termine della sezione del programma principale.

Esercizio: Come scrivere il programma CANZONE2.BAS

L'esercizio che segue descrive i passi necessari per la creazione del programma CANZONE2.BAS (figura 7-2). Può essere utile seguire questi passi ogni volta che aggiungete un nuovo sottoprogramma al programma principale.

1. Scegliete Nuovo dal menu File per iniziare un nuovo programma e digitate le righe comprese tra la numero 6 e la numero 15 della figura 7-2.
2. Posizionate il cursore sulla chiamata al sottoprogramma Ritornello (parola *Ritornello*) e selezionate Nuovo SUB dal menu Modifica per aprire la seguente finestra di dialogo:

3. Se nella finestra di dialogo appare un altro nome (oppure se non appare nulla) se ne può specificare uno digitandolo nella finestra. In questo caso il nome del sottoprogramma è già presente e perciò basta premere Invio per aprire una finestra per il nuovo sottoprogramma.

```
1   ' CANZONE2.BAS
2   ' Questo programma mostra le parole della canzone "John Brown"
3
4   DECLARE SUB Ritornello ()  'Dichiara il sottoprogramma Ritornello
5
6   CLS
7   PRINT "---------------------John Brown---------------------"
8   PRINT
9
10  PRINT "John Brown giace nella tomba là nel pian"     'Prima strofa
11  PRINT "dopo una lunga lotta contro l'oppressor."
12  PRINT "John Brown giace nella tomba là nel pian"
13  PRINT "ma l'anima vive ancor.'
14
15  Ritornello 'Chiama il programma relativo
16
17  PRINT "Con 19 suoi compagni di valor"          'Seconda strofa
18  PRINT "dall'Est all'Ovest la Virginia conquistò"
19  PRINT "con 19 suoi compagni di valor"
20  PRINT "ma l'anima vive ancor..."
21
22  Ritornello 'Chiama il programma relativo
23
24  INPUT "Premere Invio per continuare...", risposta$   'Pausa
25  PRINT
26
27  PRINT "Poi lo hanno ucciso come fosse un traditor"   'Terza strofa
28  PRINT "ma traditore fu colui che lo impiccò;"
29  PRINT "poi lo hanno ucciso come fosse un traditor"
30  PRINT "ma l'anima vive ancor..."
31
32  Ritornello 'Chiama il programma relativo
33
34  PRINT "John Brown è morto ma lo schiavo è in libertà" 'Quarta strofa
35  PRINT "tutti fratelli bianchi e neri siamo già."
36  PRINT "John Brown è morto ma lo schiavo è in libertà"
37  PRINT "ma l'anima vive ancor..."
38
39  Ritornello 'Chiama il programma relativo
40
41  END
42
43  SUB Ritornello
```

Figura 7-2. *Un programma che stampa le parole della canzone "John Brown" usando il sottoprogramma Ritornello (continua).*

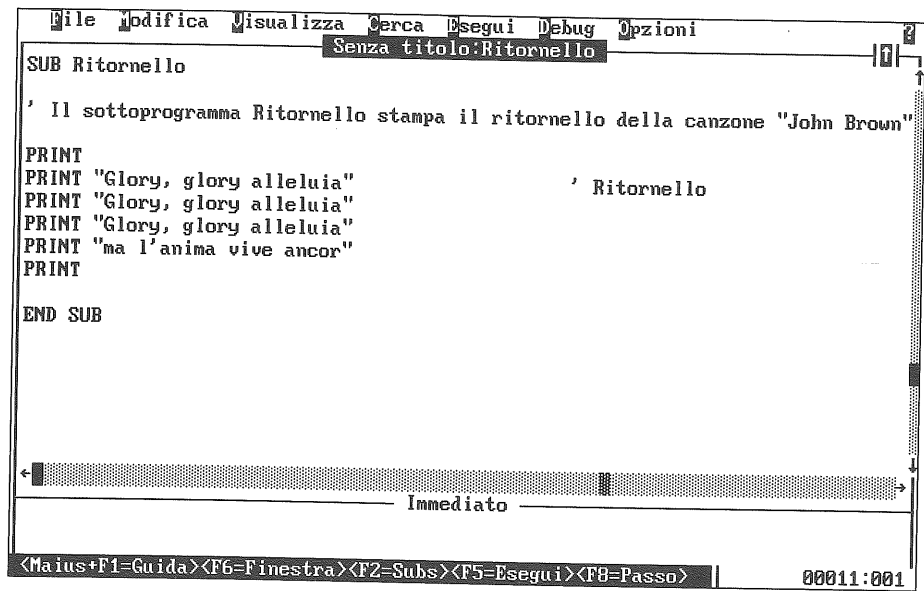
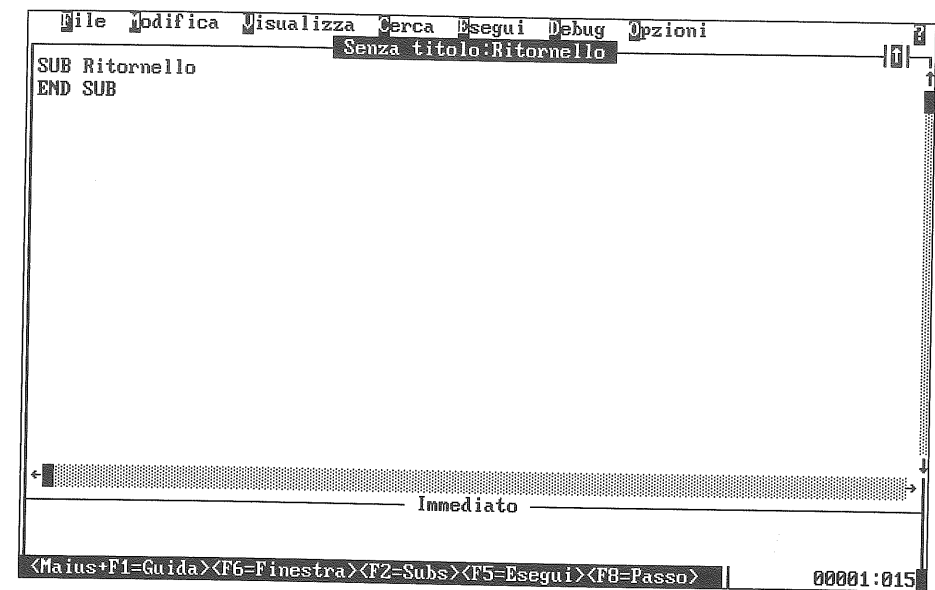
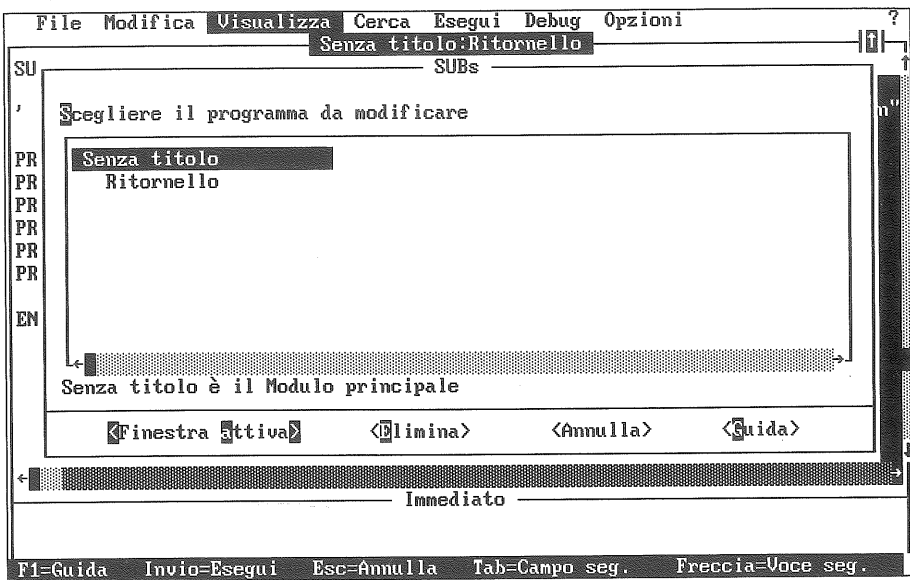

```
44
45  'Il sottoprogramma Ritornello stampa il ritornello della canzone
"John Brown"
46
46  PRINT
48  PRINT "Glory, glory alleluia"                    Ritornello
49  PRINT "Glory, glory alleluia"
50  PRINT "Glory, glory alleluia"
51  PRINT "ma l'anima vive ancor
52  PRINT
53
54  END SUB
```

Figura 7-2. *Un programma che stampa le parole della canzone "John Brown" usando il sottoprogramma Ritornello.*

```
DECLARE SUB Ritornello ()   DICHIARAZIONE DEL SOTTOPROGRAMMA
...
Ritornello   CHIAMATA AL SOTTOPROGRAMMA
...
END   FINE PROGRAMMA PRINCIPALE

SUB Ritornello   INIZIO DEL SOTTOPROGRAMMA
PRINT
PRINT "Glory, glory alleluia"  Ritornello
PRINT "Glory, glory alleluia"                 <-- CORPO DEL SOTTOPROGRAMMA
PRINT "Glory, glory alleluia"
PRINT "ma l'anima vive ancor
PRINT
END SUB   FINE DEL SOTTOPROGRAMMA
```

Figura 7-3. *Le componenti di una tipica definizione e chiamata di un sottoprogramma.*

4. Poiché non ci sono parametri da elencare si può premere Invio per iniziare una nuova riga. Si noti che la barra del titolo mostra il nome del programma e quello della procedura, separati da due punti.

5. Scrivete il corpo del sottoprogramma (righe da 44 a 53 della figura 7-2). Si noti che QBasic aggiunge automaticamente la riga 54 (l'enunciato END SUB). Il vostro schermo dovrebbe ora apparire in questo modo:

6. Selezionate il comando SUBs dal menu Visualizza per vedere una finestra di dialogo che mostra il programma principale, i sottoprogrammi e le funzioni contenute nel programma.

7. Usate i tasti di direzione per evidenziare il programma principale (al momento *Senza titolo*) e premete Invio. QBasic mostra il programma principale nella Finestra di digitazione e riposiziona il cursore sulla riga in cui lo avevate lasciato.

8. Salvate il programma col nome CANZONE2.BAS. Notate che prima di salvare il programma su disco QBasic inserisce un enunciato DECLARE in cima al programma (riga 4 della figura 7-2). A questo punto lo schermo dovrebbe apparire come riportato nella figura di pagina seguente.

9. Spostate il cursore sulla riga immediatamente sopra la chiamata del sottoprogramma Ritornello e digitate il resto del programma (righe da 16 a 41 della figura 7-2).

10. Aggiungete le righe 1, 2, 3 e 5 in cima al programma e salvatelo.

Una volta completate queste fasi eseguite il programma premendo contemporaneamente Maiusc+F5 e osservate le schermate delle parole della canzone. Il risultato dovrebbe essere identico a quello del programma CANZONE.BAS; in caso contrario confrontatelo con il listato della figura 7-2 e fate gli opportuni cambiamenti. Il programma CANZONE2.BAS è un po' più lungo di quelli visti fino ad ora per cui gli errori di battitura possono essere più difficili da trovare.

```
 File  Modifica  Visualizza  Cerca  Esegui  Debug  Opzioni                    ?
                              CANZONE2.BAS
DECLARE SUB Ritornello ()
CLS
PRINT "------------------John Brown----------------------"
PRINT

PRINT "John Brown giace nella tomba là nel pian"       ' Prima strofa
PRINT "dopo una lunga lotta contro l'oppressor."
PRINT "John Brown giace nella tomba là nel pian"
PRINT "ma l'anima vive ancor."

Ritornello        ' Chiama il programma relativo

                              Immediato

<Maius+F1=Guida><F6=Finestra><F2=Subs><F5=Esegui><F8=Passo>          00011:001
```

Questo programma dimostra una regola generale: quando i programmi sono lunghi ci vuole più tempo per renderli operativi. Non bisogna scoraggiarsi: il capitolo 12 affronterà in dettaglio il problema del debugging e della correzione dei principali errori di programmazione.

USO DI VARIABILI CON PROCEDURE

Fino ad ora abbiamo usato un ristretto gruppo di variabili; ciò non si verifica quando si scrivono programmi più complessi. Per aiutare a tenere sotto controllo grandi quantità di variabili, QBasic contiene alcune regole speciali che riguardano l'uso delle variabili all'interno delle procedure. In questa sezione vedremo queste regole sviluppando il concetto di variabili locali e globali e analizzando come passano da una procedura all'altra.

Variabili locali e globali

QBasic classifica le variabili in due gruppi: globali e locali. Una variabile globale viene dichiarata nel programma principale e cambia valore in base alle indicazioni del programma principale e delle procedure. Una variabile locale viene dichiarata nel programma principale, in un sottoprogramma o in una funzione ed è valida solo nella procedura in cui viene dichiarata; questo

tipo di variabile non risente di cambiamenti effettuati al di fuori della procedura a cui appartiene.

In questo modo è possibile usare variabili importanti in una qualunque parte del programma (globalmente) e variabili meno importanti solo all'interno di una procedura (localmente); mantenendo questa distinzione è possibile scrivere più righe di codice in forma modulare. Poiché le variabili locali sono confinate all'interno delle proprie procedure, i programmi che le usano hanno obiettivi più generali di quelli che si servono di variabili globali.

Dichiarazione di variabili globali

Per dichiarare una variabile globale si colloca un enunciato COMMON SHARED all'inizio del programma principale. Questa dichiarazione consente di usare o modificare una variabile sia all'interno del programma sia all'interno delle procedure. Questo enunciato non assegna un valore alla variabile. I valori vengono infatti assegnati successivamente per mezzo di enunciati di assegnamento. La sintassi dell'enunciato COMMON SHARED è la seguente:

```
COMMON SHARED variabileLista
```

variabileLista è una lista di una o più variabili separate da una virgola. Un enunciato COMMON SHARED può contenere un qualsiasi numero di variabili.

Esercizio: Uso di variabili globali

Il programma GLOBALE.BAS (figura 7-4) mostra come la variabile globale *gioco$* viene dichiarata nel programma principale e modificata dal sottoprogramma *NuovoGioco*.

Scrivete ed eseguite il programma GLOBALE.BAS, ottenendo questo risultato:

```
Nel programma principale, gioco$ = Scacchi
Nel sottoprogramma NuovoGioco, gioco$ = Backgammon
Di nuovo nel programma principale, gioco$ = Scacchi e Backgammon
```

Il programma mostra che qualunque cambiamento apportato alla variabile globale *gioco$* si riflette in tutto il programma.

```
' GLOBALE.BAS
' Questo programma mostra l'uso di una variabile globale

DECLARE SUB NuovoGioco ()' dichiara il sottoprogramma NuovoGioco

COMMON SHARED gioco$      ' dichiara gioco$ variabile globale

gioco$ = "Scacchi"        ' inizializza gioco$ col valore di "Scacchi"

CLS

PRINT "Nel programma principale, gioco$ = "; gioco$        ' visualizza nel
                                                           ' programma
                                                           ' principale

NuovoGioco                                  ' visualizza nel
                                            ' sottoprogramma

PRINT "Di nuovo nel programma principale, gioco$ = "; gioco$   ' visual.
                                                 ' nel programma principale

END

SUB NuovoGioco

gioco$ = gioco$ + " e Backgammon"

PRINT "Nel sottoprogramma NuovoGioco, gioco$ = "; gioco$

END SUB
```

Figura 7-4. *Un programma che mostra l'uso di una variabile globale in un programma.*

Dichiarazione di variabili locali

Le variabili sono locali per definizione il che significa che non bisogna usare alcun enunciato particolare per dichiararle. Fino a questo momento tutte le variabili dichiarate sono state variabili locali.

Esercizio: Uso di variabili locali

Il programma LOCALE.BAS (figura 7-5) rimuove l'enunciato COMMON SHARED dal programma GLOBALE.BAS per dimostrare che una variabile

locale contenuta nel programma principale e una contenuta in un sottoprogramma non interagiscono tra loro. Entrambe le variabili sono dette gioco$ ed entrambe sono variabili stringa ma, nonostante questo, sono completamente indipendenti l'una dall'altra.

Modificate il programma GLOBALE.BAS in modo che corrisponda a LOCALE.BAS ed eseguitelo. Otterrete i seguenti risultati:

```
Nel programma principale, gioco$ = Scacchi
Nel sottoprogramma NuovoGioco, gioco$ = e Backgammon
Di nuoco nel programma principale, gioco$ = Scacchi
```

Come potete vedere le modifiche apportate a *gioco$* nel sottoprogramma NuovoGioco non si riflettono sulla variabile *gioco$* contenuta nel programma principale.

```
' LOCALE.BAS
' Questo programma dimostra l'uso di una variabile locale

DECLARE SUB NuovoGioco ()       ' dichiara il sottoprogramma NuovoGioco

gioco$ = "Scacchi"        ' inizializza gioco$ col valore di "Scacchi"

CLS

PRINT "Nel programma principale, gioco$ = "; gioco$      ' visualizza nel
                                                         ' programma
                                                         ' principale

NuovoGioco   ' visualizza nel sottoprogramma

PRINT "Di nuovo nel programma principale, gioco$ = "; gioco$   ' visual.
                                                  ' nel programma principale

END

SUB NuovoGioco

gioco$ = gioco$ + " e Backgammon"

PRINT "Nel sottoprogramma NuovoGioco, gioco$ = "; gioco$

END SUB
```

Figura 7-5. *Un programma che mostra l'uso di una variabile locale in un programma.*

Trasferimento di argomenti a un sottoprogramma

È possibile accedere a un qualsiasi numero di variabili dichiarandole come globali. Per quanto conveniente, questo metodo aumenta il rischio che una variabile venga accidentalmente modificata in qualche parte del programma.

Per questa ragione QBasic fornisce uno strumento per condividere argomenti (come variabili, costanti e risultati di espressioni e funzioni) con un numero limitato di sottoprogrammi piuttosto che con tutto il programma. Questo metodo di condivisione viene detto *passaggio di argomenti*.

Gli argomenti passati a un sottoprogramma vengono ricevuti da *parametri* rappresentati da variabili locali interne al sottoprogramma; queste variabili vengono usate in modo del tutto simile alle altre.

Argomenti contro parametri

Cerchiamo di spiegare la differenza tra argomenti e parametri:

- **Un argomento è una costante, una variabile o un'espressione *passata a* un sottoprogramma.** I nomi di argomento appaiono dopo il nome del sottoprogramma nella chiamata a una procedura. Una raccolta di argomenti viene detta *elenco di argomenti.*
- **Un parametro è una variabile che riceve un valore passato a un sottoprogramma.** I parametri appaiono negli enunciati SUB e DECLARE e seguono le regole applicate a tipi di dati standard. Una raccolta di parametri viene detta *l'elenco di parametri.*

A ogni argomento deve corrispondere un parametro (non necessariamente con lo stesso nome); la figura 7-6 mostra la relazione tra un elenco di argomenti e un elenco di parametri.

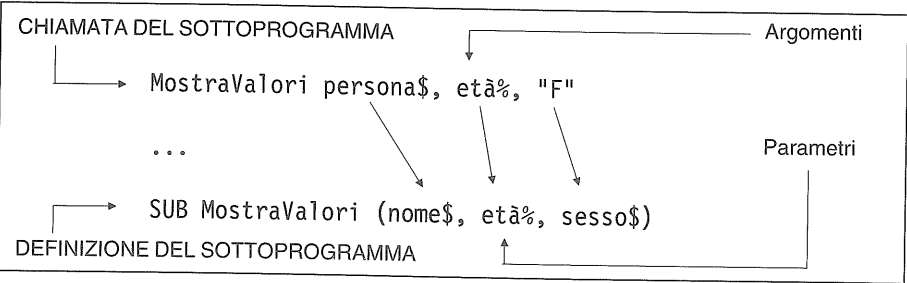



Figura 7-6. *Relazione tra argomenti e parametri.*

L'elenco che segue mostra alcune combinazioni argomento-parametro ammesse:

```
GetInput nome$, numero$, indirizzo$
...
SUB GetInput (persona$, telefono$, indirizzo$)

TradNorvegese maiale$, cane$
...
SUB TradNorvegese (gris$, hund$)

StampaIntest ORA$, DATA$, Titolo$, NumPag%
...
SUB StampaIntest (oraCorrente$, dataCorrente$, Titolo$, num%)

ChiamataMista testoInt%, testoStringa%, VALORE!, "Ciao", 747
...
SUB ChiamataMista (a%, b$, v!, d$, e%)
```

Modifica di argomenti

Il passaggio di argomenti a un sottoprogramma non avviene in un'unica direzione. Qualunque valore che un sottoprogramma assegna a un parametro viene restituito all'argomento abbinato quando il sottoprogramma è terminato.

Prendiamo come esempio la figura 7-6: se *persona$* conteneva il valore *Carla* al momento della chiamata a *MostraValori*, il parametro nome avrebbe ricevuto quel valore.

Se il sottoprogramma avesse poi assegnato il valore *Giulia* a *nome$*, questo valore sarebbe stato riassegnato a *persona$*, una volta completato il sottoprogramma.

Ricordate in ogni caso che costanti ed espressioni restano invariate; i loro valori non possono cambiare. Nella figura 7-6 per esempio, anche se il sottoprogramma *MostraValori* modificasse il valore di *sesso$*, nulla succederebbe all'argomento corrispondente *("F")*.

Esercizio: Passaggio di argomenti a un sottoprogramma

Il programma ARGOMENT.BAS (figura 7-7) mostra come due argomenti vengono passati al sottoprogramma *AggInteresse*. Il parametro *meseNome$* riceve l'argomento *mese$* mentre il parametro *totale!* riceve l'argomento *bilancio!*. Successivamente il sottoprogramma *AggInteresse* modifica il mese,

moltiplica il valore di totale! per 1.05 e restituisce entrambi i valori al programma principale.

```
' ARGOMENT.BAS
' Mostra il trasferimento di argomenti a un sottoprogramma.

DECLARE SUB AggInteresse (meseNome, totale!)

mese$ = "Gennaio"  'inizializza mese col valore di "Gennaio"

bilancio! = 1500   'inizializza bilancio! con un valore pari a 1500

CLS

PRINT "Prima del sottoprogramma:"       'visualizza valori originali
PRINT " mese$ = "; mese$
PRINT "bilancio! = "; bilancio!
PRINT

AggInteresse mese$, bilancio!           'chiama il sottoprogramma per
                                        ' modificarne i valori

PRINT "Dopo il sottoprogramma:"         'visualizza i valori modificati
PRINT "mese$ = "; mese$
PRINT "bilancio! = "; bilancio!

END

SUB AggInteresse (meseNome$, totale!)

meseNome! = "Febbraio"                  'cambia il nome del mese
totale! = totale! * 1.05                ' aggiunge all'ammontare il 5%

END SUB
```

Figura 7-7. *Un programma che mostra il passaggio di argomenti a un sottoprogramma.*

Scrivete il programma ARGOMENT.BAS ed eseguitelo; dovreste ottenere il seguente risultato:

```
Prima del sottoprogramma:
     mese$ = Gennaio
     bilancio! = 1500
```



```
Dopo il sottoprogramma:
     mese$ = Febbraio
     bilancio! = 1575
```

In questo esempio le funzioni *mese$* e *bilancio!* sono state modificate senza ricorrere a variabili globali.

CREAZIONE DI FUNZIONI

Una funzione è un blocco di codice compreso tra gli enunciati FUNCTION ed END FUNCTION. Le funzioni seguono le stesse regole dei sottoprogrammi, con la differenza che una funzione esegue un compito e restituisce un valore al programma principale o alla procedura di chiamata. Queste funzioni vengono usate in modo del tutto simile alle funzioni interne di QBasic, cioè inserendole in un enunciato. In questo paragrafo vedremo come creare delle funzioni personali (dette funzioni definite dall'utente).

Sintassi di una funzione

La sintassi di una funzione è questa:

```
FUNCTION NomeFunzione (parametroLista)
     dichiarazione di  variabile locale e di costante
     enunciati di funzione
     NomeFunzione = valoreRestituito
END FUNCTION
```

- *FUNCTION* è l'enunciato che indica l'inizio della definizione di una funzione.
- *NomeFunzione* è il nome della funzione che deve terminare con un carattere di dichiarazione del tipo (come si fa per il nome di una variabile). Il nome della funzione può avere una lunghezza massima di 40 caratteri e viene usato per chiamare la funzione. Il nome della funzione non può essere un parola chiave di QBasic o lo stesso nome di una variabile o di una procedura del programma.
- *(parametroLista)* è un elenco facoltativo di variabili
- *dichiarazione di variabile locale e di costante* è un elenco facoltativo di variabili e costanti dichiarate e usate nella funzione. Esse non hanno al-

cun effetto su variabili o costanti con lo stesso nome in altre parti del programma.

- *enunciati di funzione* è la parte operativa della funzione, nella quale si possono usare la maggior parte degli enunciati di QBasic.
- *NomeFunzione = valoreRestituito* è un enunciato di assegnazione nel corpo della funzione (generalmente alla fine) che imposta i valori restituiti dalla funzione stessa. *valoreRestituito* può essere una variabile o il risultato di un'espressione o di un'altra funzione. *NomeFunzione* e *valoreRestituito* devono essere dello stesso tipo.
- *END FUNCTION* è l'enunciato di QBasic che indica la fine della definizione di una funzione.

Per vedere come funzionano questi elementi esaminiamo la chiamata seguente, che passa tre argomenti interi e restituisce un valore intero assegnato alla variabile *numero%*:

```
numero% = SommaDiValori% (a&, b%, c%)
```

Notate che il nome della funzione *SommaDiValori%* contiene un carattere di dichiarazione del tipo intero uguale a quello nella variabile *numero%*. Una funzione definita dall'utente restituisce un valore corrispondente a uno dei cinque tipi di dati disponibili in QBasic: stringa, intero, intero lungo, in virgola mobile a precisione singola, in virgola mobile a precisione doppia.

Creazione di una funzione

La creazione di una funzione definita dall'utente è molto simile a quella di un sottoprogramma:

1. Si usa il comando Nuova FUNCTION dal menu Modifica per inserire la funzione (così come si usa Nuova SUB per inserire un nuovo sottoprogramma).
2. Si usa il comando SUBs dal menu Visualizza per esaminare e modificare le funzioni, nello stesso modo visto per i sottoprogrammi.

Esercizio: Uso di una funzione definita dall'utente

Il programma MEDIA.BAS (figura 7-8) mostra molti degli elementi di una funzione definita dall'utente nella dichiarazione della funzione *Media!*. *Me-*

dia! restituisce la media di tre parametri interi passati alla funzione stessa. Una volta eseguita la funzione, il nome *Media!* viene a rappresentare un valore in virgola mobile a precisione singola all'interno del programma principale, che può essere assegnato a una variabile in virgola mobile o visualizzato con un enunciato PRINT.

```
' MEDIA.BAS
' Questo programma stampa la media di tre numeri

DECLARE FUNCTION Media! (int1!, int2!, int3!)      'dichiara la funzione

CLS

PRINT "Inserire tre numeri di cui si vuole la media."
PRINT
INPUT "Primo intero: ", primo%
INPUT "Secondo intero: ", secondo%
INPUT "Terzo intero: ", terzo%

PRINT
PRINT "La media ™"; Media!(primo%, secondo%, terzo%)

END

FUNCTION Media! (int1%, int2%, int3%)

somma% = int1 + int2 + int3%    'somma degli argomenti
Media! = somma% /  ' restituisce un valore uguale alla media
                   ' degli argomenti

END FUNCTION
```

Figura 7-8. *Un programma che stampa il valore restituito da una funzione.*

Dopo aver scritto ed eseguito il programma MEDIA.BAS, otterrete un output simile a questo:

```
Inserire tre numeri di cui si vuole la media.
     Primo intero: 10
     Secondo intero: 20
     Terzo intero: 30
La media è 26.66667
```

Funzioni e sottoprogrammi a confronto

Le funzioni e i sottoprogrammi presentano una serie di caratteristiche comuni:

- La chiamata a una funzione contiene lo stesso numero di argomenti presenti nell'elenco di parametri della funzione stessa.
- L'elenco di parametri di una funzione riceve valori passati dal programma principale o da procedure di chiamata a cui vengono assegnati dei nomi di variabili così da poterli usare nel corpo della funzione.
- Una funzione può dichiarare e usare variabili locali e costanti.
- Una funzione viene dichiarata vicino all'inizio del programma principale con un enunciato DECLARE, generata automaticamente da QBasic.
- Una funzione viene conservata separatamente nell'ambiente QBasic ed è accessibile col comando SUBs nel menu Visualizza.
- Le funzioni vi aiutano a strutturare i programmi in moduli e a evitare la ripetizione di parti di codice.

QUALE BISOGNA USARE?

Dopo questo capitolo sarete portati a pensare che sottoprogrammi e funzioni siano essenzialmente la stessa cosa: entrambi sono strutture di programmazione flessibili che possono gestire compiti diversificati restituendo uno o più valori al modulo di chiamata. Quali sono allora le effettive differenze tra sottoprogrammi e funzioni e quale usare in un determinato contesto di programmazione?

Un sottoprogramma è un mini programma

Si può pensare a un sottoprogramma come a un programma autonomo che deve svolgere un compito importante all'interno di un programma ed essere abbastanza generico da poter essere impiegato per altri progetti. I sottoprogrammi sono spesso più lunghi delle funzioni e vengono generalmente usati quando l'informazione dev'essere visualizzata sullo schermo.
In generale, è meglio servirsi dei sottoprogrammi per eseguire compiti quali:

- Ricevere informazioni dall'utente.
- Visualizzare informazioni sullo schermo.
- Elaborare molti valori numerici o stringhe.
- Disegnare forme grafiche o disegni.
- Eseguire note musicali o canzoni.
- Restituire valori multipli ai moduli di chiamata.

Uso del comando Dividi

Il comando Dividi nel menu Visualizza permette di vedere contemporaneamente due parti distinte del programma. Ciò è particolarmente utile quando il programma contiene molti sottoprogrammi e funzioni e si vuole modificare una procedura mentre se ne controlla un'altra. Quella che segue è un elenco di passi da seguire quando si usa il comando Dividi. Poiché questo comando può essere usato in molti modi diversi, vi consiglio di fare un po' di pratica e scegliere poi la procedura che ritenete migliore.

1. Selezionando il comando Dividi dal menu Visualizza la Finestra di digitazione si divide orizzontalmente in due; la finestra superiore, che contiene il cursore, è quella attiva.
2. Per fare modifiche nella finestra superiore basta scorrere il codice fino alla parte desiderata. Se le finestre contengono la stessa parte del programma le modifiche appariranno in entrambe non appena si sposta il cursore dalla riga modificata.
3. Per attivare la finestra inferiore si usa F6 (o si fa clic col mouse); per tornare nella finestra superiore si preme F6 due volte (o si fa clic nella finestra superiore).
4. Per collocare i contenuti di una procedura in una delle due finestre basta selezionarle e premere F2 per visualizzare l'elenco delle procedure e selezionare quella da aprire.
5. Per tornare a una sola finestra si seleziona nuovamente il comando Dividi nel menu Visualizza.

Una funzione restituisce un valore

Una funzione è particolarmente utile nel calcolare e restituire al programma principale un valore singolo; le funzioni non sono adatte per restituire valori multipli o eseguire compiti di carattere generale. Ecco un elenco di compiti per cui è preferibile l'uso di una funzione:

- Eseguire un calcolo numerico.
- Restituire un valore stringa.
- Generare un numero a caso.
- Convertire un valore in un altro.
- Valutare un'espressione logica e restituire un valore vero o falso.
- Calcolare un risultato da più argomenti.

Il programma principale gestisce dichiarazioni e controlli

Quali sono i compiti del programma principale? In realtà non molti se si fa largo uso di sottoprogrammi e funzioni. Questo elenco mostra alcuni dei compiti del programma principale:

- Commenti e spiegazioni introduttive.
- Inizializzazione di variabili e strutture chiave.
- Codice da eseguire una sola volta.
- Strutture di controllo del flusso che determinano il percorso di esecuzione del programma.

Le dichiarazioni seguenti devono essere incluse nel programma principale se sono parte del programma:

- Dichiarazioni di variabili globali e costanti.
- Dichiarazioni di sottoprogrammi e funzioni.

L'uso di un programma principale, di sottoprogrammi e funzioni permette di scrivere programmi modulari e meglio organizzati.

SOMMARIO

In questo capitolo abbiamo esaminato sottoprogrammi e funzioni, che sono due strutture di programmazione che vi permettono di evitare ripetizioni. I sottoprogrammi vengonodefiniti tra gli enunciati SUB e END SUB, mentre le funzioni tra FUNCTION ed END FUNCTION; entrambe queste procedure si inseriscono, e se ne conserva traccia, con il sistema di menu di QBasic. Entrambe supportano variabili locali e scambio di argomenti dal modulo di chiamata. Sottoprogrammi e funzioni, combinati con le strutture di controllo del flusso e di looping (viste nei capitoli 5 e 6), rappresentano un *set* completo di strumenti per scrivere programmi strutturati e ben organizzati.

DOMANDE ED ESERCIZI

1. Quali, tra i seguenti, sono vantaggi che derivano dall'uso delle procedure?
   a. Le procedure possono essere chiamate un numero indefinito di volte.
   b. Le procedure permettono di creare propri enunciati e funzioni.
   c. Le procedure facilitano lo sviluppo di un programma.
   d. Le procedure generali possono essere incorporate in altri progetti di programmazione.
2. Dov'è l'errore in questo enunciato SUB?
   ```
   SUB ScriviNome$ (Nome$, Cognome$)
   ```
3. Qual è la scorciatoia di tastiera del comando SUBs nel menu Visualizza?
4. Quale comando di menu si può usare per eseguire un'intera procedura in una sola volta?
5. Scrivete un enunciato che dichiara una variabile globale dal nome *totale%*.
6. Qual è la differenza tra un sottoprogramma e una funzione?
7. Scrivete un sottoprogramma dal nome *InfoAuto* che chiede all'utente la marca, il modello, l'anno e il colore della sua auto e restituisce queste informazioni nel programma principale in quattro variabili. Una volta terminato scrivete un enunciato che chiama il sottoprogramma *InfoAuto*.

8. Scrivete una funzione dal nome *TrovaMaggiore%* che accetta due argomenti interi e restituisce quello più grande al modulo di chiamata. Una volta terminato scrivete un enunciato che chiama la funzione *TrovaMaggiore%*.

9. Scrivete un programma che usa dei sottoprogrammi per stampare una raccolta di caratteri disposti in modo da formare tre figure: una riga, un rettangolo o un triangolo.

CAPITOLO 8

Lavorare con grandi quantità di informazioni

Dopo aver esaminato l'uso di semplici tipi di dati e variabili vediamo come QBasic gestisce grandi quantità di dati all'interno di un programma. Le tecniche esposte in questo capitolo si riveleranno molto utili per creare strutture di dati (cioè raccolte di singoli elementi di informazione) che vi aiuteranno a organizzare grandi quantità di informazioni e a velocizzare operazioni che implicano molte variabili.

CONSERVAZIONE E RECUPERO DI INFORMAZIONI

Conoscete tre modi per immagazzinare dei valori all'interno di un programma:

- Assegnando un valore a una costante:

```
CONST PREZZO! = 12.950
```

- Assegnando un valore a una variabile:

```
Nome$ = "Antonio"
```

- Assegnando un valore a una variabile con l'enunciato INPUT:

```
INPUT "Scrivere il numero di casa: ", numCasa%
```

Questi sono tutti valori singoli. Cosa succede se si dispone di molti valori che si pensa di usare molte volte e in un ordine specifico? In questi casi si ricorre agli enunciati DATA e READ.

Uso degli enunciati DATA e READ

Gli enunciati DATA e READ permettono di immagazzinare e recuperare informazioni all'interno di un programma: i valori vengono prima immagazzinati in uno o più enunciati DATA e poi vengono assegnati a variabili per mezzo di uno o più enunciati READ. La sintassi di questi due enunciati è la seguente:

```
READ variabileLista
DATA costanteLista
```

variabileLista è un elenco di una o più variabili separate da una virgola; *costanteLista* è un elenco di uno o più valori numerici o stringhe separate da una virgola. Ogni valore nell'enunciato DATA deve avere una corrisponden-

te variabile di tipo appropriato nell'enunciato READ. Per esempio, quella che segue è una coppia corretta di enunciati DATA e READ. Notate come i valori appaiono nello stesso ordine delle variabili.

```
READ nome$, anni%

DATA Gianni Scotti, 25
```

L'esempio che segue mostra il funzionamento di READ e DATA: in questo programma l'enunciato READ assegna il primo valore DATA alla variabile stringa *nome$* e il secondo alla variabile intera *anni%*. L'enunciato PRINT utilizza poi le variabili.

```
READ nome$, anni%
PRINT nome$; " ha"; anni%; "anni."
DATA Gianni Scotti, 25
```

Quando il programma viene eseguito si ottiene il seguente risultato:

```
Gianni Scotti ha 25 anni
```

La figura 8-1 mostra come i valori vengono assegnati alle variabili.

```
READ nome$, anni%

PRINT nome$; " ha"; anni%; "anni."

DATA Gianni Scotti, 25
```

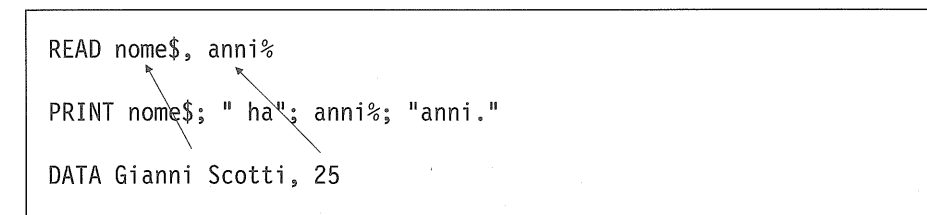



Figura 8-1. *I valori conservati nell'enunciato DATA vengono assegnati alle variabili con un enunciato READ.*

Affermazioni DATA e READ: suggerimenti

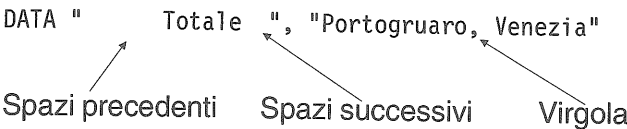
Nell'usare gli enunciati DATA e READ per immagazzinare e recuperare informazioni è bene tenere a mente i seguenti punti:

- È possibile creare più enunciati DATA e READ, ma bisogna assicurarsi che ogni variabile READ abbia un corrispondente valore DATA:

```
READ mese$, numeroDiGiorni%
READ vacanze%

DATA Novembre
DATA 30, 3
```

- Se un valore nell'enunciato DATA contiene una virgola, un segno di due punti o un certo numero di spazi, l'intero valore dev'essere racchiuso tra virgolette:

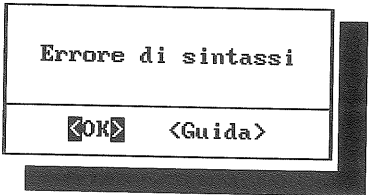
```
DATA "       Totale  ", "Portogruaro, Venezia"
```

Spazi precedenti   Spazi successivi   Virgola

- Gli enunciati DATA devono comparire nel programma principale (vengono generalmente collocate alla fine). Gli enunciati READ devono essere collocati sopra quelli DATA nel programma principale ma, all'interno delle procedure possono apparire ovunque.

Cosa succede se i tipi di dati non sono uguali?

Ogni valore contenuto in un enunciato DATA deve essere assegnato a una corrispondente variabile dello stesso tipo in un enunciato READ. Se i tipi non corrispondono, QBasic effettua le seguenti conversioni:

Se il valore DATA è	*Ma la variabile READ è*	*Il risultato della conversione è*
stringa	intero/virgola mobile	*errore di sintassi*
intero/virgola mobile	stringa	stringa
intero	virgola mobile	virgola mobile
virgola mobile	intero	arrotondato all'intero

Un errore di sintassi viene indicato da questa finestra di dialogo:

Il programma che segue mostra una discrepanza di tipo tra il valore DATA (una stringa) e la variabile READ (un intero).

```
READ nome$
PRINT "Nome: "; nome$
READ indirizzo$
PRINT "Indirizzo: "; indirizzo$
READ anni%
PRINT "Età: "; anni%
DATA Gianni Scotti, "Via Roma 9, Ravenna", nove
```

Al momento dell'esecuzione la schermata di output mostra brevemente questo risultato:

```
Nome: Gianni Scotti
Indirizzo: Via Roma 9, Ravenna
```

Subito dopo appare il messaggio *errore di sintassi* perché l'enunciato READ non può assegnare un valore stringa (*nove*) a una variabile intera (*anni%*). Per correggere il programma basta cambiare *nove* con *9*, perché concordi con la variabile intera *anni%* o cambiare *anni%* con *anni$* perché concordi con il valore stringa *nove*.

Valori tipici degli enunciati DATA

L'uso degli enunciati DATA e READ come sistema di immagazzinamento e recupero di valori è particolarmente indicato quando si conosce il valore delle variabili in anticipo e si sa che i valori appariranno sempre nello stesso ordine. Ecco un elenco di valori per i quali valgono queste condizioni:

- Giorni della settimana (da Lunedì a Domenica).
- Mesi dell'anno (da Gennaio a Dicembre).
- Nomi di persone, posti o organizzazioni.
- Dati numerici per calcoli o analisi.
- Valori numerici per note musicali.

Esercizio: Immagazzinare molti valori

Il programma AGGIUNGI.BAS (figura 8-2) mostra come usare un enunciato READ all'interno di un loop FOR per assegnare più valori DATA a delle variabili. Si noti che la costante intera *ELEMENTI%* controlla il numero di elementi letti dagli enunciati DATA; modificando *ELEMENTI%* è perciò possibile cambiare il numero di elementi da leggere.

1. Scrivete ed eseguite il programma AGGIUNGI.BAS

```
' AGGIUNGI.BAS
' Questo programma legge e aggiunge i valori conservati
' in enunciati DATA.

CONST ELEMENTI% = 20      'Imposta il numero di oggetti da leggere

CLS

FOR i% = 1 TO ELEMENTI%  'per ogni elemento da leggere
READ numero! 'assegna il successivo elemento DATA a numero!
somma! = somma! + numero!'aggiunge l'elemento al totale corrente
NEXT i%

PRINT "La somma di"; ELEMENTI%; "numeri è"; somma!

DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
DATA 88.2, 25, 3.3, 100, -74.2, 0, 20, 0.34, -89, 5.4567
```

Figura 8-2. *Un programma che usa READ per assegnare 20 elementi alle variabili.*

L'output del programma mostrerà il seguente risultato:

```
La somma di 20 numeri è 134.0967
```

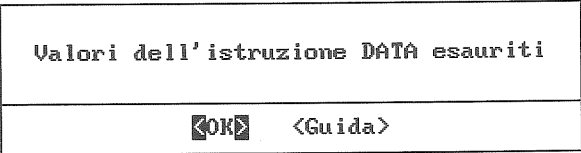
2. Cambiate il valore della costante ELEMENTI% a 4 ed eseguite di nuovo il programma; avrete sullo schermo il seguente risultato:

```
La somma di 4 numeri è 10
```

L'indicatore di fine dati

Se il programma non è in grado di identificare l'ultimo valore DATA continua ad eseguire gli enunciati READ (cercando di assegnare un valore a una

variabile). Il risultato che ne consegue è un messaggio di errore che indica che nessun dato è disponibile per l'assegnazione:

```
Valori dell'istruzione DATA esauriti

<OK>   <Guida>
```

Per evitare questo errore inserite un *indicatore di fine dati* come ultima voce dell'ultimo enunciato DATA; Il programma si serve di tale valore per determinare il punto in cui terminano i valori DATA. Quando il programma legge l'*identificatore*, passa a eseguire il restante codice. Questa tecnica è utile quando si dispone di una lunga lista di valori all'interno di enunciati DATA che verrà elaborata solo una volta. Il programma AGGIUNGI.BAS aggira questo potenziale problema perché usa una costante che contiene un numero di valori che corrisponde esattamente al limite superiore del ciclo FOR. Non potrete godere spesso di un simile lusso! Nell'esercizio che segue AGGIUNGI.BAS viene modificato perché possa leggere un indicatore di fine dati.

Controllare i valori DATA: il puntatore di dati

Per aiutare l'enunciato READ ad assegnare i valori alle variabili, QBasic usa un *puntatore ai dati* per indicare il valore del successivo enunciato DATA da assegnare. Quando un programma inizia, il puntatore indica il primo valore nel primo enunciato DATA; a mano a mano che l'esecuzione prosegue e i valori DATA vengono assegnati, il puntatore indica i successivi valori ancora da assegnare.

Esercizio: Inserimento di un indicatore di fine dati

Il programma AGGIUNG2.BAS (figura 8-3) usa un indicatore di fine dati (–9999) come ultima voce dell'enunciato DATA nel programma. Scrivete ed eseguite il programma AGGIUNG2.BAS:

```
' AGGIUNG2.BAS
' Questo programma legge e aggiunge i valori conservati
' in un enunciato DATA fino a che incontra un indicatore
' di fine dati (-9999).

CLS

DO WHILE (numero! <> -9999)      ' esegue un loop fino a che incontra
                                 'l'indicatore di fine dati
     READ numero! ' assegna il successivo elemento DATA a numero!
     IF (numero! <> -9999) THEN      ' se non incontra l'indicatore
                                     ' di fine dati allora
          somma! = somma! + numero! ' Conserva un totale corrente
          elementi% = elementi% + 1 ' Calcola il numero di valori
                                    ' letti
     END IF
LOOP

PRINT "La somma di"; elementi%; "numeri è"; somma!

DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
DATA 88.2, 25, 3.3, 100, -74.2, 0, 20, 0. 34, -89, 5.4567
```

Figura 8-3. *Un programma che legge i dati fino al raggiungimento dell'indicatore di fine dati.*

Lo schermo dovrebbe mostrare il seguente risultato:

```
La somma di 20 numeri è 134.0967
```

Rileggere i valori DATA con l'enunciato RESTORE

A volte può essere necessario leggere ripetutamente la lista di valori DATA in un programma: per esempio potreste voler eseguire una serie di calcoli diversi su uno stesso gruppo di valori. Per ritornare al primo dei valori DATA servitevi dell'enunciato RESTORE che reimposta il puntatore dei dati sul primo valore dell'enunciato DATA contenuto nel programma. RESTORE può essere usato un numero indefinito di volte.

La sintassi dell'enunciato RESTORE:

```
RESTORE
```


Esercizio: Uso di RESTORE per ripetere un elenco di valori.

Il programma ORETV.BAS (figura 8-4) usa RESTORE, DATA e READ per controllare quante ore una persona passa davanti al televisore durante un periodo di tre settimane. Una coppia di loop FOR nidificati scorre per tre volte i giorni della settimana (per simulare tre settimane) registrando il numero totale di ore di visione. L'enunciato RESTORE collocato alla fine di ogni ciclo riporta il puntatore al primo giorno (in questo caso domenica).

NOTA: Per vivacizzare il risultato sullo schermo si è usato l'enunciato COLOR che visualizza il numero totale di ore trascorse davanti al video in rosso lampeggiante; naturalmente potrete vedere questo effetto solo se disponete di un monitor a colori.

1. Scrivete ed eseguite il programma ORETV.BAS; alla fine riceverete tre richieste di informazioni simili a queste:

```
Quante ore avete passato davanti alla TV nelle ultime 3
settimane?
Domenica, Settimana1 --> 2
Lunedi', Settimana 1 --> 0
Martedi', Settimana 1 --> 1
Mercoledi', Settimana 1 --> 0
Giovedi', Settimana 1 --> 2
Venerdi', Settimana 1 --> 0
Sabato, Settimana 1 --> 2.5
```

2. Rispondete alle domande e, dopo aver completato i 21 valori, lo schermo output mostrerà il risultato finale in questa forma:

```
Avete guardato 23.5 ore di televisione nelle ultime tre
settimane!
```

Dopo aver imparato a immagazzinare e recuperare grandi quantità di valori esaminiamo i modi più efficienti per la loro gestione. Il paragrafo che segue introduce la *matrice* (array), un potente strumento di strutturazione dei dati che può aiutare a gestire grandi quantità di informazioni dello stesso tipo.

LE MATRICI

Le matrici servono per organizzare sotto uno stesso nome variabili dello stesso tipo.

```
' ORETV.BAS
' Questo programma usa DATA, READ e RESTORE per controllare il numero
' di ore trascorse davanti alla TV in un periodo di tre settimane.

CLS

PRINT "Quante ore avete passato davanti alla TV nelle ultime 3
settimane?
PRINT

FOR i% = 1 TO 3     ' Per ognuna delle ultime tre settimane
      FOR j% = 1 TO 7     ' e per ogni giorno nella settimana
            READ giorno$ ' leggi il nome del giorno dalla lista DATA
            PRINT giorno$; ", Settimana"; i%;        ' Richiedi il giorno
                                                     ' e la settimana
            INPUT "--> ", ore!          ' Recupera il numero di ore per
                                        ' quel giorno
            totaleOre! = totaleOre! + ore! ' Totale di tutte le ore
      NEXT j%
      PRINT ' Stampa una riga vuota dopo ogni settimana
      RESTORE      ' Sposta il puntatore nuovamente sulla Domenica
NEXT i%

PRINT "Avete guardato"    ' Mostra il numero totale di ore
COLOR 20                  ' in colore rosso lampeggiante
PRINT totaleOre!; "ore";
COLOR 7                   ' Ristabilisce il bianco come colore del testo
PRINT "di televisione nelle ultime tre settimane!"

DATA Domenica, Lunedi', Martedi', Mercoledi', Giovedi', Venerdi',
Sabato
```

Figura 8-4. *Un programma che mostra l'uso dell'enunciato RESTORE.*

Una matrice può contenere tutti i tipi di dati visti sino ad ora per cui si potranno avere:

- Matrici di dati stringa.
- Matrici di dati interi.
- Matrici di dati interi lunghi.
- Matrici di dati in virgola mobile a precisione singola.

- Matrici di dati in virgola mobile a precisione doppia.

Non è possibile mescolare tipi diversi all'interno di una singola matrice (una matrice di stringhe può contenere solo stringhe e così via).

Vedremo un esempio che organizza i dati in tre tipi di matrici: matrici di stringhe, matrici di interi e matrici di dati in virgola mobile a precisione singola.

Raccogliere informazioni con le matrici: il Mercato Della Bicicletta

Il Mercato Della Bicicletta, un grosso negozio del centro, ha sette commessi. Lorenzo, il proprietario vuole registrare mensilmente due valori:

- Numero di biciclette vendute da ogni commesso.
- Numero di lire fatturate da ogni commesso.

Viene naturale organizzare queste informazioni in tre elenchi, come mostra la figura 8-5:

Commesso	Biciclette vendute	Fatturato
Fabio	6	2.600.000
Andrea	5	3.000.000
Sandro	12	4.600.000
Emanuela	7	2.400.000
Annalisa	11	5.200.000
Paola	2	1.000.000
Valerio	5	2.500.000

Figura 8-5. *Un esempio dei dati di vendita del Mercato Della Bicicletta.*

- Una lista di valori stringa (i nomi dei commessi).
- Una lista di valori interi (il numero di biciclette vendute per persona).
- Una lista di valori in virgola mobile (il totale fatturato per ogni commesso).

È evidente come Lorenzo nel redigere questi elenchi abbia già fatto uso di matrici. Addirittura di tre: una matrice di stringhe per i nomi delle persone, una matrice di interi per il numero di biciclette vendute e una matrice di dati

in virgola mobile a precisione singola per il denaro fatturato. Ogni elenco, con un suo differente tipo di informazione, si qualifica come una matrice che contiene sette *elementi* o valori individuali. Dopo aver organizzato le matrici su carta le si può convertire in un programma con l'aiuto dell'enunciato DIM.

L'enunciato DIM: prenotare la matrice

Quando si prenota un tavolo a un ristorante si forniscono delle informazioni in cambio della prenotazione: il nome, il tipo di pasto (colazione, pranzo o cena) e il numero dei convitati. Nel mondo delle matrici l'enunciato DIM svolge la stessa funzione di una prenotazione, fornendo lo spazio di memoria necessario per contenerla; in cambio richiede tre informazioni:

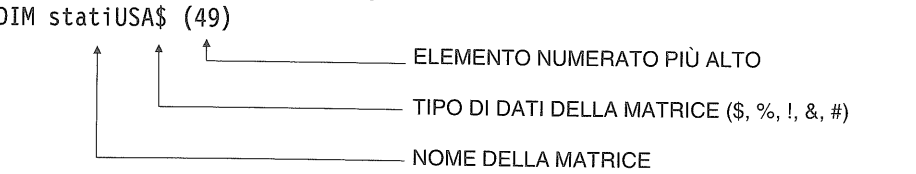
- Il nome scelto per la matrice.
- Il tipo di dati che si intende inserirvi.
- Il massimo numero di elementi che la matrice può contenere.

Questo processo di allocazione dello spazio viene detto *dimensionamento*. La sintassi dell'enunciato DIM è la seguente:

```
DIM nomeMatrice (coordinata)
```

nomeMatrice è il nome scelto per la matrice e *coordinata* è il più alto elemento numerato che vi è contenuto. Il carattere finale di *nomeMatrice* dev'essere il carattere di dichiarazione del tipo che identifica il tipo di dati contenuti nella matrice: *$* per stringhe, *%* per interi, *&* per interi lunghi, *!* per numeri decimali (in virgola mobile) a precisione singola o *#* per numeri decimali (in virgola mobile) a precisione doppia.

L'enunciato DIM di questo esempio dimensiona una matrice di stringhe che può contenere fino a 50 elementi numerati da 0 a 49:

```
DIM statiUSA$ (49)
```

Al momento dell'esecuzione di quest'enunciato DIM, QBasic riserva memoria per una matrice di 50 elementi che contiene i nomi degli stati americani.

NOTA: In genere, un computer ha spazio per molte matrici, contenenti anche migliaia di elementi. Tuttavia, poiché la memoria non è illimitata, è bene evitare di riservare spazio in più di quello strettamente necessario.

Lorenzo potrebbe creare le tre matrici di cui ha bisogno usando questi enunciati DIM:

- Per i nomi dei commessi una matrice di stringhe di sette elementi detta *commessi$*:

  ```
  DIM commessi$(6)
  ```

- Per il numero di biciclette vendute un matrice di interi di sette elementi detta *biciVendute%*:

  ```
  DIM biciVendute%(6)
  ```

- Per il totale del fatturato una matrice di dati in virgola mobile a precisione singola di sette elementi detta *totVendite!*

  ```
  DIM totVendite!(6)
  ```

Notate che in tutte e tre le matrici il numero 6 riserva lo spazio di memoria necessario per sette elementi numerati da 0 a 6.

Ogni elemento della matrice è associato a un numero. Per definizione QBasic associa il primo elemento di una matrice col numero 0, il secondo col numero 1 e così via per gli altri, come si vede nella figura 8-6.

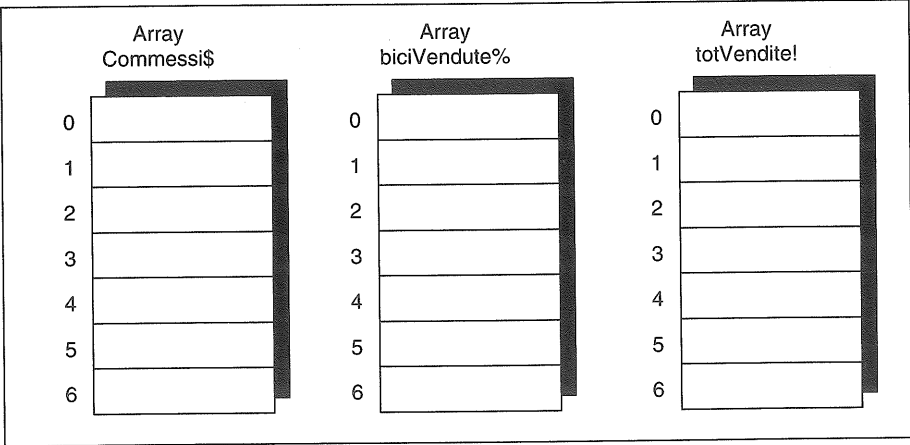



Figura 8-6. *Ogni elemento di una matrice dimensionata è associato a un numero.*

Lavorare con gli elementi di una matrice

Dopo aver dimensionato una matrice con l'enunciato DIM, è molto semplice far riferimento agli elementi in essa contenuti. Per indicare un elemento di una matrice si usa il nome della matrice e un *indice* di matrice racchiuso tra parentesi. L'indice deve essere un valore intero: un numero semplice o una variabile intera. L'enunciato seguente assegna il valore stringa *Patate* all'elemento 3 nella matrice *listaSpesa$*:

```
listaSpesa$(3) = "Patate"
```

L'enunciato OPTION BASE

Se volete che il primo elemento di una matrice corrisponda a 1 invece che a 0 potete usare l'enunciato OPTION BASE; l'enunciato associa il primo elemento (base) di tutte le matrici contenute in un programma, al numero 1. Per usare OPTION BASE dovete semplicemente collocarlo all'inizio del programma prima di qualsiasi enunciato DIM:

```
OPTION BASE 1
```

Nei programmi che seguono, questo enunciato verrà usato nel modo ora descritto.

Esercizio: Conservazione di valori in una matrice

Il programma REGNOMI.BAS (figura 8-7) mostra come le informazioni vengono conservate in una matrice e stampate usando due loop FOR. La costante PERSONE% contiene il numero dei commessi del negozio di biciclette e OPTION BASE fa corrispondere il primo elemento di ogni matrice a 1 invece che a 0.

Scrivete ed eseguite il programma REGNOMI.BAS. Il programma richiede di fornire i nomi dei commessi e produce un risultato simile a questo:

```
Inserire il nome del commesso:Fabio
Inserire il nome del commesso:Andrea
Inserire il nome del commesso:Sandro
Inserire il nome del commesso:Emanuela
Inserire il nome del commesso:Annalisa
Inserire il nome del commesso:Paola
Inserire il nome del commesso:Valerio
```



```
' REGNOMI.BAS
' Questo programma legge delle informazioni stringa all'interno
' di una matrice

CONST PERSONE% = 7          ' Numero di commessi
OPTION BASE 1               ' Imposta la base di tutte le matrici a 1

DIM Commessi$(PERSONE%)     ' Dimensione della matrice di stringhe
                            ' Commessi$
CLS

FOR i% = 1 TO PERSONE%      ' Usa i% per accedere agli elementi della
                            ' matrice
      INPUT "Inserire il nome del commesso: ", Commessi$(i%)
NEXT i%

PRINT
PRINT "Avete inserito i nomi seguenti: "
PRINT

FOR i% = 1 TO PERSONE%      ' Stampa il contenuto complessivo della
                            ' matrice
      PRINT Commessi$(1%)
NEXT i%
```

Figura 8-7. *Un programma che mostra il caricamento e la stampa di una matrice.*

```
Avete inserito i nomi seguenti:
Fabio
Andrea
Sandro
Emanuela
Annalisa
Paola
Valerio
```

Formattazione del risultato: l'enunciato PRINT USING

L'enunciato PRINT USING consente di definire come apparirà il risultato sullo schermo ed è particolarmente utile se si vogliono visualizzare grandi quantità di dati in forma tabulare. Per creare il formato desiderato si usa una stringa modello. La sintassi dell'enunciato PRINT USING è la seguente:

```
PRINT USING modello; argomentoLista
```

modello è una stringa e *argomentoLista* è una raccolta di uno o più valori da visualizzare, separati da punti e virgola; *modello* serve a specificare come dovrebbero essere visualizzati i valori contenuti in *argomentoLista*. I caratteri di formattazione contenuti in *modello* devono corrispondere ai caratteri dei valori contenuti in *argomentoLista*. La tabella seguente descrive alcuni utili caratteri di formattazione che possono essere inclusi in un modello:

Carattere(i)	Descrizione
#	Rappresenta una cifra di un valore numerico
.	Rappresenta il punto decimale in un valore numerico
$$	Visualizza un simbolo di dollaro assieme al numero
\\	Rappresenta uno o più spazi che possono essere riempiti con dati stringa

Il programma seguente crea un modello di formattazione detto *tmp$* e lo usa in un enunciato PRINT USING per visualizzare una stringa di variabili e un valore in virgola mobile a precisione singola (dollari e centesimi):

```
tmp$ = "Nome:  \                               \ Pagamento: $$####.##"
nome$ = "Gigi Poli"
pagamento! = 2496.33
PRINT USING tmp$; nome$; pagamento!
```

L'esecuzione dovrebbe dare questo risultato:

```
Nome: Gigi Poli                             Pagamento: $2496.33
```


Esercizio: Uso di più matrici in un programma

Il programma INFOBICI.BAS (figura 8-8) mostra come usare più matrici per raccogliere informazioni correlate. INFOBICI è una semplice revisione di REGNOMI.BAS in cui i nomi dei commessi vengono registrati con la matrice *commessi$*, il numero di biciclette vendute da ognuno con la matrice *biciVendute%* e il fatturato di vendita di ciascuna persona con la matrice *totVendite%*. Le tre matrici vengono progettate per essere usate insieme (ognuna infatti contiene un'informazione sui commessi del negozio di biciclette, come si vede nella figura 8-6). Se fate riferimento all'insieme degli indici di matrice in un loop FOR, il programma potrà accedere nello stesso momento agli elementi correlati. L'enunciato PRINT USING e il modello *tmp$* visualizzano i dati contenuti in ciascuna matrice.

Scrivete ed eseguite il programma INFOBICI.BAS.

```
' INFOBICI.BAS
' Questo programma legge e stampa le informazioni di tre matrici.
CONST PERSONE% = 7 ' numero di commessi
OPTION BASE 1      ' imposta la base di tutte le matrici a 1
DIM Commessi$(PERSONE%)  ' dimensiona la matrice di stringhe Commessi$
DIM biciVendute%(PERSONE%)     ' dimensiona la matrice di interi
                               ' biciVendute%
DIM totVendite!(PERSONE%)      ' dimensiona la matrice di punti mobili
                               ' totVendite!
CLS
FOR i% = 1 TO PERSONE%         ' recupera i nomi e i dati di vendita
                               ' dei commessi
      INPUT "inserire il nome del commesso: ", Commessi$(i%)
      INPUT "Biciclette vendute: ", biciVendute%(i%)
      INPUT "Totale vendite: L ", totVendite!(i%)
PRINT
NEXT i%
PRINT "Avete inserito i seguenti dati di vendita:"
PRINT
PRINT "Commesso   Bici vendute Totale vendite"
PRINT "----------------------------------------------------------"
PRINT
' Inizializzazione di tmp$, modello di formattazione per PRINT USING
tmp$ =       "\      \ ###  L$#.###.###
FOR i% = 1 TO PERSONE%   ' stampa i contenuti di ogni matrice
      PRINT USING tmp$; Commessi$(i%); biciVendute%(i%); totVendite!(i%)
NEXT i%
```

Figura 8-8. *Uso di tre matrici in un loop FOR.*

L'output del programma dovrebbe essere simile a questo:

```
Inserire il nome del commesso: Fabio
      Biciclette vendute: 6
      Totale vendite: L 2.600.000

Inserire il nome del commesso: Andrea
      Biciclette vendute: 5
      Totale vendite: L 3.000.000

Inserire il nome del commesso: Sandro
      Biciclette vendute: 12
      Totale vendite: L 4.600.000

Inserire il nome del commesso: Emanuela
      Biciclette vendute: 7
      Totale vendite: L 2.400.000

Inserire il nome del commesso: Annalisa
      Biciclette vendute: 11
      Totale vendite: L 5.200.000

Inserire il nome del commesso: Paola
      Biciclette vendute: 2
      Totale vendite: L 1.500.000

Inserire il nome del commesso: Valerio
      Biciclette vendute: 5
      Totale vendite: L 2.500.000

Avete inserito i seguenti dati di vendita:

Commesso    Bici vendute     Totale vendite
-------------------------------------------------------------------
Fabio              6          2.600.000
Andrea             5          3.000.000
Sandro            12          4.600.000
Emanuela           7          2.400.000
Annalisa          11          5.200.000
Paola              2          1.500.000
Valerio            5          2.500.000
```


Riempire parte di una matrice

Non è necessario riempire una matrice completamente; anzi può essere una buona idea lasciare spazio per dati da inserire in futuro. In questo caso dovrete trovare un modo per indicare a QBasic che l'utente ha terminato l'inserimento dei dati. Una soluzione è quella di inserire un indicatore di fine dati nel modo visto in precedenza. Potete fare in modo che l'utente aggiunga degli elementi alla matrice nell'ambito di un loop WHILE che controlla in continuazione l'esistenza dell'indicatore di fine dati. Non appena QBasic legge l'indicatore esce dal loop ed esegue il resto del programma. L'indicatore dovrebbe corrispondere al tipo di dati contenuti dalla matrice e dovrebbe avere un valore che sia improbabile che l'utente digiti durante la normale esecuzione del programma. Un tipico indicatore di fine dati per una matrice di stringhe è ESCI o FINE, per una matrice numerica è -9999.

Esercizio: Uso di un indicatore di fine dati

Il programma FINEDATI.BAS (figura 8-9) mostra come riempire una matrice con diverse quantità di dati. Il programma riempie e stampa tre matrici che, questa volta, vengono dimensionate per contenere 50 elementi ciascuna e usa un indicatore di fine dati (FINE) per indicare che l'utente ha terminato l'inserimento dei dati (si noti che FINE deve essere in maiuscolo). Scrivete ed eseguite il programma FINEDATI.BAS.

Il risultato dovrebbe essere simile a questo:

```
Inserire i dati su richiesta. Digitare FINE per uscire

Inserire il nome del commesso: Fabio
      Biciclette vendute: 6
      Totale vendite: L 2.600.000

Inserire il nome del commesso: Emanuela
      Biciclette vendute: 7
      Totale vendite: L 2.400.000

Inserire il nome del commesso: FINE
Avete inserito i seguenti dati di vendita:

Commesso    Bici vendute     Totale vendite
-------------------------------------------
Fabio             6          2.600.000
Emanuela          7          2.400.000
```

```
' FINEDATI.BAS
' Legge e stampa le informazioni contenute in tre matrici.
' Il numero massimo di nomi inseribili è 50: se ne possono inserire
' di meno digitando "FINE" al posto del nome del commesso.

OPTION BASE 1       ' imposta la base di tutte le matrici a 1

DIM Commessi$(50)          ' dimensiona la matrice di stringhe Commessi$
DIM biciVendute%(50)       ' dimensiona la matrice di interi biciVendute%
DIM totVendite!(50)        ' dimensiona la matrice di punti mobili
                           ' totVendite!

CLS

PRINT "Inserire i dati su richiesta. Digitare FINE per uscire."
PRINT

calcolo% = 1 ' inizializza una variabile per il calcolo della matrice

WHILE (Commessi$(calcolo%0 <> "FINE") 'continua finché nome = "FINE"
      INPUT "Inserire il nome del commesso: ", Commessi$(calcolo%)
      IF (Commessi$(calcolo%) <> "FINE") THEN
            INPUT " Biciclette vendute: ", biciVendute%(calcolo%)
            INPUT " Totale vendite: L ", totVendite!(calcolo%)
            PRINT
            calcolo% = calcolo% + 1   ' incremento nel calcolo delle
                                      ' matrici
      END IF
WEND

PRINT
PRINT "Avete inserito i seguenti dati di vendita:"
PRINT
PRINT "Commesso    Bici vendute Totale vendite"
PRINT "-------------------------------"
PRINT

' inizializzazione di tmp$, modello di formattazione per PRINT USING
tmp$ =       "\     \ ### L$#.###.###
FOR i% = 1 TO PERSONE%    ' Stampa i contenuti di ogni matrice
      PRINT USING tmp$; Commessi$(i%); biciVendute%(i%); totVendite!(i%)
NEXT i%
```

Figura 8-9. *Un programma che usa un indicatore di fine dati per determinare quando un input è terminato.*

Creazione di matrici flessibili

Quando si usa l'enunciato DIM bisogna informare QBasic del numero di elementi nella matrice per poter riservare uno spazio adeguato in memoria. Cosa succede se non si è sicuri del numero di elementi che la matrice conterrà? Per esempio, se il programma dipende dall'input dell'utente per quanto concerne i contenuti della matrice, il loro numero potrà variare moltissimo ogni volta che il programma viene eseguito. Come si può allora indicare a QBasic la quantità di memoria da riservare alla matrice?

L'enunciato DIM permette di creare due tipi di matrici a seconda del tipo di informazione fornita: matrici statiche e matrici dinamiche.

Matrici statiche

Una matrice statica presenta una dimensione definita (cioè conosciuta in anticipo). Per esempio, nel programma INFOBICI.BAS il numero dei commessi è conosciuto in anticipo e perciò viene collocato come costante:

```
CONST PERSONE% = 7
```

L'enunciato DIM usa la costante PERSONE% come numero degli elementi contenuti nella matrice:

```
DIM Commessi$(PERSONE%)
```

Matrici dinamiche

Una matrice dinamica è una matrice la cui dimensione può cambiare. Una matrice dinamica viene dimensionata solo quando l'utente del programma definisce il numero di elementi che la matrice dovrà contenere. Per creare una matrice dinamica si dovrebbero seguire questi passi:

1. Usare un enunciato INPUT per richiedere all'utente il numero di elementi necessari.
2. Assegnare il valore inserito dall'utente a una variabile intera.
3. Usare la variabile intera con l'enunciato DIM per dimensionare così la matrice.

Si consiglia di usare una matrice statica quando se ne conosce la dimensione in anticipo e una matrice dinamica quando la dimensione verrà determinata ogni volta che il programma viene eseguito.

Esercizio: Uso di una variabile per impostare la dimensione della matrice

Il programma DINAMIC.BAS (figura 8-10) usa la variabile *persone%* per dimensionare le tre matrici dinamiche delle vendite. Si noti come l'enunciato IF controlli il valore di *persone%*: se questo è inferiore o uguale a 10 le matrici non vengono dimensionate. Scrivete ed eseguite il programma DINAMIC.BAS.

Il risultato ottenuto sarà simile a questo:

```
Quanti nomi di commessi vuoi inserire? 2

Inserire il nome del commesso: Fabio
Biciclette vendute: 6
Totale vendite: L 2.600.000

Inserire il nome del commesso: Emanuela
Biciclette vendute: 7
Totale vendite: L 2.400.000

Avete inserito i seguenti dati di vendita:

Commesso    Bici vendute    Totale vendite
-----------------------------------------------------------------
Fabio            6           2.600.000
Emanuela         7           2.400.000
```

Ricerca di un elemento nella matrice

Molto spesso è necessario effettuare delle ricerche all'interno della matrice per trovare un elemento specifico oppure per trovare un elemento sulla base di un confronto. Per eseguire entrambe queste operazioni è necessario che il programma esamini ogni singolo elemento della matrice e tenga traccia di quelli che corrispondono alla stringa di ricerca.

- Nel cercare un elemento specifico QBasic confronta la stringa di ricerca con ogni elemento della matrice finché non trova un elemento uguale o finché tutti gli elementi non sono stati esaminati.


```
' DINAMIC.BAS
' Legge e stampa le informazioni in tre matrici dinamiche

OPTION BASE 1       ' imposta la base di tutte le matrici a 1

CLS

INPUT "Quanti nomi di commessi vuoi inserire? ", persone%
IF (persone% > 0) THEN   ' bisogna inserire almeno una persona

DIM Commessi$(persone%)   ' dimensiona la matrice di stringhe Commessi$
DIM biciVendute$(persone%)       ' dimensiona la matrice di interi
                                 ' biciVendute%
DIM totVendite!(persone%)        ' dimensiona la matrice in virgola mobile
                                 ' totVendite!

PRINT

FOR i% = 1 TO persone%    ' raccoglie nome e dati di vendita dei commessi
     INPUT "Inserire il nome del commesso: ", Commessi$(i%)
     INPUT " Biciclette vendute: ", biciVendute%(i%)
     INPUT " Totale vendite: L ", totVendite!(i%)
     PRINT
NEXT i%

PRINT "Avete inserito i seguenti dati di vendita:"
PRINT
PRINT "Commesso    Bici vendute Totale vendite"
PRINT "-------------------------------------------------------------"
PRINT

' Inizializzazione di tmp$, modello di formattazione per PRINT USING
tmp$ =      "\     \ ###  L$#.###.###

FOR i% = 1 TO PERSONE%    ' stampa i contenuti di ogni matrice
     PRINT USING tmp$; Commessi$(i%); biciVendute%(i%); totVendite!(i%)

NEXT i%

END IF
```

Figura 8-10. *Un programma che dimostra l'uso di tre matrici dinamiche.*

- Nell'effettuare una ricerca basata sul confronto si usano una o più variabili temporanee per registrare il progredire del confronto. In genere i confronti assumono le forme seguenti:
- Trova il numero più grande contenuto nella matrice.
- Trova il numero più piccolo contenuto nella matrice.

Gli esercizi che seguono mostrano alcuni metodi tipici di ricerca di un elemento specifico e del numero maggiore contenuto in una matrice.

Esercizio: Ricerca di un elemento della matrice

Il programma RICERCA.BAS (figura 8-11) mostra come cercare un elemento specifico all'interno di una matrice. Il programma richiede all'utente alcuni dati sul commesso e quali informazioni si vogliono esaminare. Un loop FOR esamina ciascun elemento della matrice *Commessi$* fino a quando non trova l'elemento cercato o fino a quando l'esame non è stato completato:

- Se la ricerca va a buon fine, QBasic mostra gli elementi corrispondenti contenuti nelle matrici *biciVendute%* e *totVendite!* per poi uscire dal loop grazie a un enunciato EXIT FOR.

L'enunciato EXIT FOR

L'enunciato EXIT FOR permette di uscire in anticipo da un loop FOR ed è utile quando si vuole che un ciclo si ripeta un certo numero di volte a meno che non si verifichi una certa condizione. Il seguente programma continua a invitare l'utente a inserire nomi fino a un massimo di dieci o finché l'utente non avrà digitato ESCI (qualunque sia la prima condizione a verificarsi).

```
FOR i% = 1 TO 10
      INPUT "Scrivi un nome: ", Nome$
      IF (Nome$ = "ESCI"") THEN EXIT FOR
      PRINT Nome$
NEXT i%
```

- Se la ricerca non va a buon fine, QBasic mostra il messaggio *Nome non trovato.*

Scrivete ed eseguite il programma RICERCA.BAS.


```
' RICERCA.BAS
' Questo programma legge i dati contenuti in tre matrici e cerca un nome.
' Il numero massimo di nomi che possono essere inseriti è 50: per
' uscire prima bisogna scrivere FINE al posto del nome del commesso.

OPTION BASE 1  ' imposta la base di tutte le matrici a 1

DIM Commessi$(50)  ' dimensiona la matrice di stringhe Commessi$
DIM biciVendute$(50)   ' dimensiona la matrice di interi biciVendute%
DIM totVendite!(50)    ' dimensiona la matrice punto mobile totVendite!
CLS
PRINT "Inserire i dati richiesti. Digitare FINE per uscire"
PRINT
calcolo% = 1    ' inizializza la variabile di calcolo di una matrice
WHILE (Commessi$(calcolo%) <> "FINE") ' continua finché nome = "FINE"
    INPUT "Inserire il nome del commesso: ", Commessi$(calcolo%)
    IF (Commessi$(calcolo%) <> "FINE") THEN
        INPUT "  Biciclette vendute:  ", biciVendute%(calcolo%)
        INPUT "  Totale vendite:  L ", totVendite!(calcolo%)
        PRINT
        calcolo% = calcolo% + 1 ' incrementa calcolo della matrice
    END IF
WEND
PRINT   ' richiede all'utente la stringa da ricercare
INPUT "Qual è il nome da cercare? ", ricerca$
PRINT

' inizializzazione di tmp$, modello di formattazione per PRINT USING
tmp$ = "\  \ ###   L$#.###.###
' confronta ogni elemento della matrice con la stringa fino a trovarne
' una uguale; poi visualizza il registro corrispondente ed esce dal loop;
' mostra il messaggio solo se la stringa viene trovata.
FOR i% = 1 TO calcolo% - 1 ' calcolo% - 1 è l'ultimo elemento della
                  ' matrice
    IF (Commessi$(i%) = ricerca$) THEN
        PRINT  "Commesso  Bici vendute           Totale vendite"
        PRINT  "----------------------------------------------"
        PRINT
        PRINT USING tmp$; Commessi$(i%); biciVendute%(i%); totVendite!(i%)
        EXIT FOR    ' questo enunciato interrompe il loop FOR
    END IF
    IF (i% = calcolo% - 1) THEN PRINT "**Nome non trovato**"
NEXT i%
```

Figura 8-11. *Un programma che mostra come cercare un elemento specifico in una matrice.*

Il risultato sarà simile a questo:

```
Inserire i dati richiesti. Digitare FINE per uscire

Inserire il nome del commesso: Fabio
Biciclette vendute: 6
Totale vendite: L 2.600.000

Inserire il nome del commesso: Emanuela
Biciclette vendute: 7
Totale vendite: L 2.400.000

Inserire il nome del commesso: Andrea
Biciclette vendute: 5
Totale vendite: L 3.000.000

Inserire il nome del commesso: FINE

Qual è il nome da cercare? Andrea

Commesso    Bici vendute     Totale vendite
-----------------------------------------------------------------
Andrea           5          3.000.000
```

Esercizio: Ricerca del numero con valore più alto contenuto in una matrice

Il programma MAXVEND.BAS (figura 8-12) mostra come estrarre l'elemento più grande contenuto in una matrice usando un loop di tipo FOR. Il programma chiede all'utente i dati sui commessi e poi esamina gli elementi della matrice *totVendite!*; il valore più alto viene conservato nella variabile *maxVend!* e poi confrontato con ogni elemento di *totVendite!* Se un altro elemento della matrice è maggiore di *maxVend!* questa variabile incorpora il nuovo elemento, scartando il precedente (la variabile *lg%* conserva l'indice della matrice associato a *maxVend!*).

Dopo aver esaminato tutti gli elementi il programma mostra il valore di vendita più alto e gli elementi correlati nelle matrici *Commessi$* e *biciVendute%*.

Scrivete ed eseguite il programma MAXVEND.BAS.

Il risultato che si ottiene dovrebbe essere simile a questo:

```
Inserire i dati richiesti. Digitare FINE per uscire

Inserire il nome del commesso: Fabio
```



```
' MAXVEND.BAS
' Questo programma legge i dati sui commessi in tre matrici e mostra i
' dati della persona col piñ alto valore totale di vendita. Numero max
' di nomi inseribili è 50; digitare "FINE" per terminare in anticipo.
OPTION BASE 1  'imposta la base di tutte le matrici a 1
DIM Commessi$(50)  ' dimensiona la matrice di stringhe Commessi$
DIM biciVendute$(50)   ' dimensiona la matrice di interi biciVendute%
DIM totVendite!(50)  ' dimensiona la matrice in virgola mobile totVendite!
CLS
PRINT "Inserire i dati richiesti. Digitare FINE per uscire"
PRINT
calcolo% = 1   ' inizializza la variabile di calcolo di una matrice
WHILE (Commessi$(calcolo%) <> "FINE") ' continua finché nome = "FINE"
    INPUT "Inserire il nome del commesso: ", Commessi$(calcolo%)
    IF (Commessi$(calcolo%) <> "FINE") THEN
        INPUT "  Biciclette vendute:  ", biciVendute%(calcolo%)
        INPUT "  Totale vendite:  L ", totVendite!(calcolo%)
        PRINT
        calcolo% = calcolo% + 1 ' incrementa il calcolo della matrice
    END IF
WEND
massimo! = totVendite!(1) ' l'elemento della prima matrice è per ora
                          ' il maggiore
lg% = 1 ' salva l'indice della matrice
' controlla gli altri elementi della matrice alla ricerca
' di un valore maggiore. Se ne trova uno lo assegna a massimo! e salva
' l'indice della matrice in lg%; se due valori sono uguali restituisce
' il primo valore trovato.
FOR i% = 2 TO calcolo% - 1
    IF (totVendite!(i%) > massimo!) THEN
        massimo! = totVendite!(i%) ' salva il nuovo valore maggiore
        lg% = i%    ' salva l'indice della matrice
    END IF
NEXT i%
' inizializzazione di tmp$, modello di formattazione per PRINT USING
tmp$ = "\  \ ###  L$#.###.###
PRINT
PRINT "** "; Commessi$(lg%); " ha il totale di vendita più elevato **"
PRINT
PRINT  "Commesso  Bici vendute                    Totale vendite"
PRINT  "-----------------------------------------------------------"
PRINT
PRINT USING tmp$; Commessi$(i%); biciVendute%(i%); totVendite!(i%)
```

Figura 8-12. *Un programma che dimostra l'estrazione dell'elemento con valore più alto in una matrice.*

```
Biciclette vendute: 6
Totale vendite: L 2.600.000

Inserire il nome del commesso: Emanuela
Biciclette vendute: 7
Totale vendite: L 2.400.000

Inserire il nome del commesso: Andrea
Biciclette vendute: 5
Totale vendite: L 3.000.000

Inserire il nome del commesso: Annalisa
Biciclette vendute: 11
Totale vendite: L 5.200.000

Inserire il nome del commesso: FINE

** Annalisa ha il totale di vendita più elevato **

Commesso    Bici vendute    Totale vendite
-----------------------------------------------------------------
Annalisa         11           5.200.000
```

Matrici bidimensionali

QBasic permette di dichiarare matrici bidimensionali, cioè matrici che rappresentano una tabella di valori con righe e colonne (come una scacchiera, un libro mastro etc.). In questa sezione vedremo come dichiarare e usare matrici bidimensionali all'interno di un programma.

Partiamo con un esempio: Carlo, un distributore di bevande, vuole registrare l'andamento delle vendite delle sue quattro marche principali negli ultimi dodici mesi. Egli vuole inserire le informazioni in una tabella suddivisa per marche e mesi, come mostrato nella figura 8-13.

Informazioni di carattere tabulare sono particolarmente adatte per matrici bidimensionali. In questo esempio, una dimensione della matrice corrisponde alle righe delle marche di bibite, l'altra alle colonne con i mesi. Gli elementi di una matrice bidimensionale vengono identificati tramite coordinate di riga e colonna; la figura 8-14 mostra come le coordinate di riga e colonna verrebbero assegnate a ciascuna dimensione se la base della matrice fosse impostata a 1. Una matrice monodimensionale richiede una sola coordinata per identificare un elemento, mentre ne sono necessari due per svolgere la stessa funzio-

	Gen	Feb	Mar	Apr	Mag	Giu	Lug	Ago	Set	Ott	Nov	Dic
Coca Spray	64	63	58	45	36	32	41	39	50	67	69	103
Fizzy	35	41	60	57	38	29	25	19	26	37	43	36
Alki	15	9	12	21	24	32	46	42	37	22	18	13
Schpritz	30	30	30	35	42	44	49	48	38	35	31	30

Figura 8-13. *Una tabella che mostra i dati di vendita negli ultimi 12 mesi.*

Uso di matrici con procedure

Nel capitolo 7 abbiamo visto come dichiarare variabili locali e globali e come gli argomenti delle variabili vengano trasferiti ai sottoprogrammi e alle funzioni. Regole simili stanno alla base dell'uso delle matrici in programmi che contengono procedure.

- Per definizione una matrice è legata al programma principale o alla procedura in cui è stata dichiarata.
- Per attribuire a una matrice carattere globale si usa l'enunciato COMMON SHARED nel programma principale; per esempio:

```
DIM negozi$(20)
COMMON SHARED negozi$()
```

- Un elemento di una matrice può essere trasferito a una procedura; per esempio:

```
GetInput negozi$(12)
```

- Una matrice intera può essere trasferita a una procedura senza specificare alcuna coordinata; per esempio:

```
StampaNegozi negozi$()
```

In programmi di grosse dimensioni l'uso di matrici insieme a sottoprogrammi e funzioni è particolarmente utile per l'organizzazione del programma.

ne in una matrice bidimensionale. Nella figura 8-14 per esempio, il numero delle casse di Coca Spray vendute in maggio sarebbe identificato dalle coordinate riga 1, colonna 5.

COORDINATE COLONNA ↓ (colonna 5)

	1	2	3	4	5	6	7	8	9	10	11	12
1 Coca Spray	64	63	58	45	36	32	41	39	50	67	69	103
2 Fizzy	35	41	60	57	38	29	25	19	26	37	43	36
3 Alki	15	9	12	21	24	32	46	42	37	22	18	13
4 Schpritz	30	30	30	35	42	44	49	48	38	35	31	30

↑ COORDINATE RIGA (riga 1)

Figura 8-14. *Assegnazione delle coordinate a una matrice bidimensionale.*

Dichiarazione di una matrice bidimensionale

Per dimensionare una matrice bidimensionale si usa l'enunciato DIM nel modo seguente:

```
DIM nomeMatrice(righe, colonne)
```

nomeMatrice è il nome della matrice, *righe* è il numero di riga (la prima dimensione) e *colonne* è il numero di colonna (seconda dimensione); *righe* e *colonne* devono essere interi e possono espressi come numeri, costanti o variabili. Il carattere finale di *nomeMatrice* dev'essere il carattere di dichiarazione del tipo che identifica il tipo di dati della matrice: *$* per stringhe, *%* per interi, *&* per interi lunghi, *!* per virgola mobili a precisione singola o *#* per virgola mobile a precisione doppia. Come in una matrice monodimensionale ogni elemento contenuto nella matrice dev'essere dello stesso tipo.

NOTA: Il primo elemento di entrambe le dimensioni della matrice viene indicato con il numero 0 a meno che non si usi l'enunciato OPTION BASE per numerarlo diversamente.

Gli enunciati seguenti dimensionano la matrice di 4 righe e 12 colonne *totBibite%*:

```
OPTION BASE 1
DIM totBibite%(4,12)
```


Questi altri enunciati invitano l'utente a inserire le dimensioni della matrice dinamica bidimensionale *totBibite%*:

```
OPTION BASE 1
INPUT "Inserire il numero delle marche vendute: ", marche%
INPUT "Inserire il numero dei mesi da registrare: ", mesi%
DIM totBibite%(marche%, mesi%)
```

Esercizio: Costruzione di una tabella di valori

Il programma VENDITE.BAS (figura 8-15) mostra come viene riempita e visualizzata sullo schermo la matrice bidimensionale *totBibite%*. VENDITE riceve le coordinate di riga e colonna dall'utente, le conserva nelle variabili intere *marche%* e *mesi%* e le usa per dimensionare la matrice *totBibite%*. VENDITE usa due loop FOR nidificati per riempire e mostrare la matrice e gli enunciati DATA e READ per conservare e recuperare i mesi dell'anno. Questo programma può essere usato come guida per riempire e stampare qualsiasi matrice dinamica bidimensionale. Scrivete il programma VENDITE.BAS.

Al momento dell'esecuzione VENDITE.BAS richiede le informazioni:

```
** Programma per la registrazione delle vendite di bibite **

Quanti tipi di bibite vendete? 4
Quanti mesi volete registrare (1-12)? 12
Inserire le 4 marche di bibite che vendete

Marca: Coca Spray
Marca: Fizzy
Marca: Alki
Marca: Schpritz

Inserire i dati di vendita delle bibite

** Coca Spray **

Gen: 64
Feb: 63
Mar: 58
...
```

Dopo aver inserito i dati relativi a tutte le marche e a tutti i mesi si dovrebbe ottenere un risultato simile al seguente:

```
E.BAS
' Registra le vendite di bibite nell'arco di un certo periodo.
CLS
PRINT "** Programma per la registrazione delle vendite di bibite **"
PRINT
DO
    INPUT "Quanti tipi di bibite vendete? ", marche%
LOOP WHILE (marche% < 1)
DO
    INPUT "Quanti mesi volete registrare (1-12)? ", mesi%
LOOP WHILE (mesi% < 1) OR (mesi% > 12)
PRINT
OPTION BASE 1  ' imposta il primo elemento della matrice a 1
DIM totBibite%(marche%, mesi%) ' dimensiona la matrice delle vendite
DIM marcaNome$(marche%)   ' dimensiona la matrice delle marche
' riceve i nomi delle marche di bibite vendute
PRINT "Inserire le"; marche%; "marche di bibite che vendete"
PRINT
FOR i% = 1 TO marche%
    INPUT "Marca: ", marcaNome$(i%)
NEXT i%
' riceve le vendite mensili delle bibite
PRINT
PRINT "Inserire i dati di vendita delle bibite"
PRINT
FOR i% = 1 TO marche%  ' per ogni marca di bibite...
    PRINT "** "; marcaNome$(i%); " **"
    PRINT   ' stampa il nome della marca
    FOR j% = 1 TO mesi%    ' per ogni mese...
        READ me$    ' Leggi il nome del mese dalla lista DATA
        PRINT "  "; me$;   ' stampa il nome del mese e richiede
                           ' informazioni
        INPUT ": ", totBibite%(i%, j%) ' conserva le informazioni
                                       ' nella matrice
    NEXT j%
    PRINT
    RESTORE ' ripercorre la lista DATA fino al primo mese
NEXT i%
' Stampa la tabella coi dati di vendita
CLS
PRINT "Vendite di bibite per"; mesi%; "mesi"
PRINT
PRINT "-----------------------------------------------------------------"
```

Figura 8-15. *VENDITE.BAS: un programma che utilizza una matrice bidimensionale dinamica per verificare le vendite delle bibite (continua).*


```
PRINT "Marca/Mese  ";
FOR i% = TO mesi%
    PRINT "  ";
    READ me$    ' Legge il nome del mese dalla lista DATA
    PRINT me$; ' stampa i nomi dei mesi in cima alla tabella
NEXT i%
PRINT
PRINT "-----------------------------------------------------------------"
' Modelli per PRINT USING
nomeMod$ = "\  \"  ' per il nome di marca (fino a 12 cifre)
venditeMod$ = " ###"   ' per le bibite vendute (fino a 3 cifre)
FOR i% = 1 TO marche%   ' Riempie la tabella
    PRINT USING nomeMod$; marcaNome$(i%)
    FOR j% = 1 TO mesi%
        PRINT USING venditeMod$; totBibite%(i%, j%);
    NEXT j%
    PRINT
NEXT i%
PRINT "-----------------------------------------------------------------"
' informazioni per gli enunciati READ
DATA Gen, Feb, Mar, Apr, Mag, Giu, Lug, Ago, Set, Ott, Nov, Dic
```

Figura 8-15. *VENDITE.BAS: un programma che utilizza una matrice bidimensionale dinamica per verificare le vendite delle bibite.*

```
Vendite di bibite per 12 mesi

-----------------------------------------------------------------------
Marca/Mese    Gen  Feb  mar  Apr  Mag  Giu  Lug  Ago  Set  Ott  Nov  Dic
-----------------------------------------------------------------------
Coca Spray     64   63   58   45   36   32   41   39   50   67   69  103
Fizzy          35   41   60   57   38   29   25   19   26   37   43   36
Alki           15    9   12   21   24   32   46   42   37   22   18   13
Schpritz       30   30   30   35   42   44   49   48   38   35   31   30
-----------------------------------------------------------------------
```

NOTA: Per stampare una copia della tabella finale su carta assicuratevi che la stampante sia accesa e poi tenete premuto il tasto Maiusc e il tasto Stampa Schermo, che serve a mandare in stampa il contenuto dello schermo.

Errori tipici dell'uso delle matrici

Se da un lato le matrici sono di grandissima utilità, dall'altro possono aumentare la probabilità di commettere degli errori. In questa sezione vedremo alcuni degli errori di programmazione tipici, associati all'uso di matrici e all'elaborazione di grandi quantità di dati.

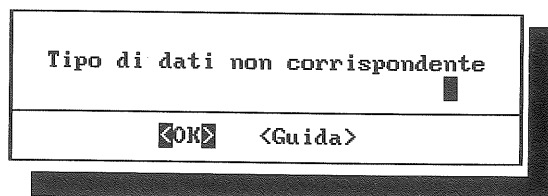
Errore 1: Non usare un intero per definire una coordinata

Il numero di elementi e dimensioni in una matrice è determinato dalle coordinate contenute nell'enunciato DIM; queste coordinate devono essere dei valori interi o delle variabili. La seguente dichiarazione di matrice è un esempio valido:

```
DIM voti%(studenti%, 15)
```

Le coordinate *studenti%* e *15* sono dei valori interi. Quest'altra dichiarazione invece non è valida perché *oggetti$* e *quantità$* sono valori stringa:

```
DIM costoTavolo!(oggetti$, quantità$)
```

Quando si prova a dimensionare una matrice che contiene delle coordinate non valide si ottiene il seguente messaggio di errore:

Se una coordinata è rappresentata da un numero in virgola mobile QBasic arrotonda il valore all'intero più vicino e poi dimensiona la matrice. Per esempio, l'enunciato seguente dimensiona una matrice con sei elementi (da 0 a 5):

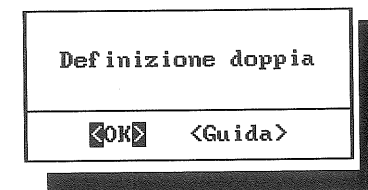
```
DIM valori% (4,6)
```

NOTA: Usare un valore a virgola mobile come coordinata di una matrice può confondere chi legge il vostro programma ed è perciò consigliabile mantenere sempre valori interi.

Errore 2: Usare un enunciato DIM in un loop

Assicuratevi che gli enunciati DIM non si trovino all'interno di loop. Questa parte di programma mostra un uso scorretto di un enunciato DIM (in corsivo nel listato) nella dichiarazione di una matrice dinamica:

```
OPTION BASE 1
dimens% = 5
CLS
FOR i% = 1 TO dimens%
      DIM nomi$(dimens%)
      INPUT "Inserite un nome: ", nomi$(i%)
NEXT i%
PRINT
FOR i% = 1 TO dimens%
      PRINT nomi$(i%)
NEXT i%
```

Se provate ad eseguire questo programma avrete un messaggio di errore al momento in cui l'enunciato DIM viene eseguito per la seconda volta:

Per evitare questo errore bisogna dimensionare la matrice *nomi$* prima del loop:

```
OPTION BASE 1
dimens% = 5
DIM nomi$(dimens%)
CLS
FOR i% = 1 TO dimens%
      INPUT "Inserire un nome: ", nomi$(i%)
NEXT i%
PRINT
FOR i% = 1 TO dimens%
      PRINT nomi$(i%)
NEXT i%
```

Errore 3: Confondere indice e valore della matrice

È facile confondere il valore usato per indicizzare un elemento della matrice con il valore corrispondente a quell'elemento. Ricordate che l'indice della matrice è sempre tra parentesi e segue il nome della matrice che descrive la collocazione del valore (figura 8-16).

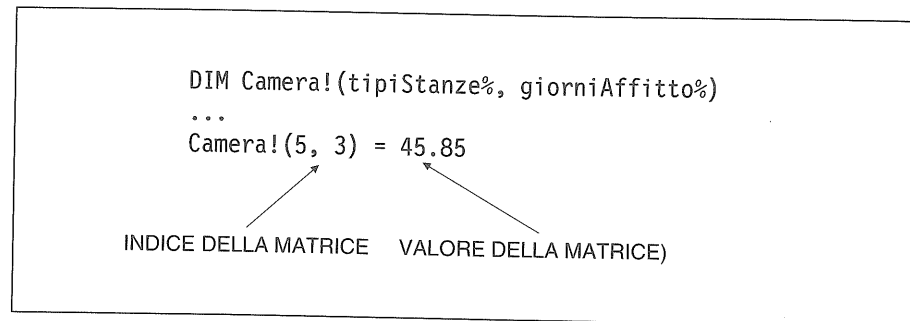



Figura 8-16. *Indice e valore di una matrice.*

L'enunciato seguente cerca di assegnare il valore stringa "Hotel Sport" al quinto elemento della matrice *hotelLiberi$* ma non ci riesce perché l'indice e il valore della matrice si trovano nella collocazione sbagliata:

```
hotelLiberi$("Hotel Sport") = 5
```

Questa invece è l'assegnazione corretta:

```
hotelLiberi$(5) = "Hotel Sport"
```

Errore 4: Discordanza tra i tipi della matrice

Ogni elemento di una matrice dev'essere dello stesso tipo; se si prova ad assegnare una variabile o un valore diverso dal tipo di matrice specificato nell'enunciato DIM, si ottiene un messaggio di errore. Per esempio il seguente enunciato di assegnazione genera un messaggio di errore perché *costo!* è una matrice in virgola mobile a precisione singola e il valore *"57.36"* è un tipo di informazione stringa (mentre l'assegnazione corretta sarebbe *57.36* senza le virgolette).

```
costo!(i%) = "57.36"
```

Questo tipo di errore di assegnazione produce il seguente messaggio:

Un analogo errore di discordanza tra tipi può ricorrere in fase di esecuzione del programma se l'utente inserisce un valore diverso dall'elemento della matrice che lo riceve. Il loop che segue, per esempio, richiede all'utente di inserire una serie di interi.

```
DIM valori%(5)
FOR i% = 1 TO 5
     INPUT "Inserire un numero: ', valori%(i%)
NEXT i%
```

Se l'utente inserisce un valore stringa ottiene il seguente messaggio:

```
Inserire un numero: dieci
Ricominciare da capo
Inserire un numero:
```

Ricominciare da capo è il messaggio con cui QBasic chiede all'utente di inserire il tipo di dati corretto. Per evitare questo genere di problemi in fase di immissione di dati è consigliabile specificare sempre, nella stringa dell'enunciato INPUT, il tipo di dati che l'utente deve inserire.

Errore 5: Valori al di fuori dell'intervallo ammesso

Se provate a far riferimento a un elemento che non esiste nella matrice, otterrete un messaggio di errore *Indice inferiore fuori limite* al momento dell'esecuzione del programma. Nell'esempio che segue QBasic genera un messaggio di errore quando viene eseguita il terzo enunciato perché *Punteggio%* può contenere solo 25 elementi e si fa riferimento al trentesimo.

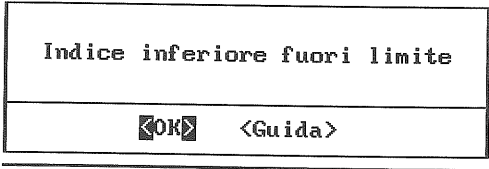
```
OPTION BASE 1
DIM Punteggio%(25)
Punteggio%(30) = 92
```

Questo tipo di errore è abbastanza comune in strutture di loop, come mostra questo esempio:

```
OPTION BASE 1
DIM colori$(4)
calcolo% = 0
CLS
DO
      calcolo% = calcolo% + 1
      INPUT "Inserire il nome di un colore: ", colori$(calcolo%)
LOOP UNTIL (colori$(calcolo%) = "FINE")
```

Al momento dell'esecuzione del programma si ottiene un risultato simile a questo:

```
 un colore: Giallo
Inserire un colore: Rosso
Inserire un colore: Blu
Inserire un colore: Verde
Inserire un colore: Bianco
```

Dopo l'inserimento del quinto colore (che si colloca nell'ultimo elemento disponibile nella matrice) il contatore del loop *calcolo%* viene aumentato e l'enunciato INPUT (che ora si riferisce a un elemento oltre i limiti della matrice) attiva il messaggio di errore *Indice inferiore fuori limite*.

La soluzione per questo tipo di problema è quella di determinare i limiti superiori e inferiori della matrice per non superarli. QBasic dispone delle funzioni UBOUND e LBOUND, le quali sono in grado di fornire i limiti di ogni matrice contenuta nel programma così da poter gestire questo problema potenziale.

Le funzioni UBOUND e LBOUND

Le funzioni UBOUND e LBOUND restituiscono dei valori interi corrispondenti ai limiti superiori e inferiori di una matrice. La sintassi di queste due funzioni è la seguente:

```
UBOUND(nomeMatrice, dimensione)
LBOUND(nomeMatrice, dimensione)
```

nomeMatrice è il nome della matrice di cui si vogliono determinare i limiti; dimensione è la dimensione che si vuole controllare (non richiesta se state prendendo in considerazione una matrice monodimensionale). Entrambe le funzioni restituiscono valori che possono essere assegnati a variabili o usati in espressioni.

Questa parte di programma usa le funzioni UBOUND e LBOUND per controllare la matrice monodimensionale *giocatori$*.

```
OPTION BASE 1
DIM giocatori$(9)
PRINT "Il limite superiore è"; UBOUND(giocatori$)
PRINT "Il limite inferiore è"; LBOUND(giocatori$)
```

Al momento dell'esecuzione si ottiene questo risultato:

```
Il limite superiore è 9
Il limite inferiore è 1
```

Se si vogliono controllare i limiti di una matrice bidimensionale bisogna fornire un numero di dimensione: 1 per la prima e 2 per la seconda. Il programma che segue usa UBOUND e LBOUND per controllare i limiti di entrambe le dimensioni in una matrice bidimensionale detta *venditeGen%*:

```
OPTION BASE 1
DIM venditeGen%(3, 4)
PRINT "Il limite superiore della prima dimensione è";
UBOUND(venditeGen%, 1)
PRINT "Il limite inferiore della prima dimensione è";
LBOUND(venditeGen%, 1)
PRINT "Il limite superiore della seconda dimensione è";
UBOUND(venditeGen%, 2)
PRINT "Il limite superiore della seconda dimensione è";
LBOUND(venditeGen%, 2)
```

Il risultato di questa porzione di programma potrebbe essere simile a questo:

```
Il limite superiore della prima dimensione è 3
Il limite inferiore della prima dimensione è 1
Il limite superiore della seconda dimensione è 4
Il limite superiore della seconda dimensione è 1
```

Esercizio: Evitare il messaggio di errore "Indice inferiore fuori limite"

BOUND.BAS (figura 8-17) mostra come usare UBOUND e LBOUND per controllare i limiti di una matrice in un loop DO evitando errori dovuti all'uscita dall'intervallo della matrice. Il programma dimensiona una matrice stringa monodimensionale detta *festa$* che contiene idee per giochi da fare in compagnia. BOUND.BAS riempie e stampa la matrice.

NOTA: Il loop DO usa l'enunciato EXIT DO, che fa uscire dal loop se l'utente digita FINE prima che tutti gli elementi siano stati specificati. EXIT DO può essere usato tutte le volte che avete bisogno di uscire da un loop DO prima che si verifichino le condizioni WHILE o UNTIL.

Scrivete ed eseguite il programma BOUND.BAS.

Il risultato ottenuto al momento dell'esecuzione dovrebbe essere simile a questo:

```
Inserire idee su giochi; digitare FINE per uscire

Gioco per feste: Alce Rossa
Gioco per feste: Guardia e Ladri
Gioco per feste: Bandiera
Gioco per feste: Nascondino
Gioco per feste: Quattro cantoni

Avete inserito i seguenti giochi:

Alce Rossa
Guardia e Ladri
Bandiera
Nascondino
Quattro cantoni
```



```
' BOUND.BAS
' Questo programma carica i dati in una matrice fino a che si
' digita FINE o se i limiti della matrice vengono superati.

OPTION BASE 1   ' imposta la base della matrice a 1
DIM festa$(5)   ' dimensiona una matrice con 5 elementi

calcolo% = 1    ' inizializza il calcolatore del loop partendo da 1

CLS ' Mostra il messaggio di introduzione e la nota su FINE
PRINT "Inserire idee su giochi; digitare FINE per uscire"
PRINT
' esegue il loop fino a che calcolo% si trova nei limiti della matrice
DO WHILE (calcolo% >= LBOUND(festa$)) AND (calcolo% <= UBOUND(festa$))
    INPUT "Gioco per feste: ", festa$(calcolo%)     ' trasferisce le
                                                    ' informazioni nella
                                                    ' matrice
    IF (festa$(calcolo%) = "FINE") THEN EXIT DO     ' se l'utente digita
                                                    ' FINE,
    calcolo% = calcolo% + 1                         ' il locp viene
                                                    ' interrotto
LOOP

PRINT
PRINT "Avete inserito i seguenti giochi:"
PRINT

FOR i% = 1 TO calcolo% - 1 ' Stampa i contenuti della matrice%
    PRINT festa$(i%)    ' calcola% - 1 contiene l'ultimo valore inserito

NEXT i%
```

Figura 8-17. *Un programma che usa LBOUND e UBOUND per evitare riferimenti a elementi della matrice esterni ai suoi limiti.*

SOMMARIO

In questo capitolo abbiamo visto alcune importanti strutture, funzioni, enunciati e tecniche che QBasic fornisce per lavorare con grandi quantità di dati:

- Uso di DATA, READ e RESTORE per assegnare a una variabile un'informazione immagazzinata in un programma.
- Creazione e uso di una matrice monodimensionale.
- Ricerca di elementi in una matrice.
- Formattazione tabulare di informazioni con PRINT USING.
- Creazione e uso di una matrice bidimensionale.
- Risoluzione di errori tipici dovuti all'uso di matrici.

Nel capitolo successivo vedremo le funzioni e le tecniche di QBasic correlate specificatamente all'uso di stringhe.

DOMANDE ED ESERCIZI

1. In un programma compare prima l'enunciato DATA o l'enunciato READ?
2. Quale tipo di informazione è particolarmente adatta per l'uso di DATA, READ e RESTORE?
3. Scrivete un programma che mostra i valori contenuti nella seguente enunciato DATA con un loop di tipo FOR:

   ```
   DATA, Gianni, Maria, Carlo, Anna, Giuseppe, Valerio, Bertilla
   ```

4. Vero o Falso: Una matrice può conservare dati di tipi diversi.
5. Scrivete un enunciato che riserva memoria per una matrice di 100 valori in virgola mobile a precisione singola.
6. Cos'è l'indicatore di fine dati?

7. Scrivete un programma che dichiara, riempie e stampa una matrice dinamica monodimensionale contenente i personaggi dei vostri film preferiti.
8. Che cosa è l' errore *Indice inferiore fuori limite*?

CAPITOLO 9

Lavorare con le stringhe

Nei capitoli 3 e 4 abbiamo visto i tipi di dati stringa e gli esercizi sulla creazione di stringhe e sulla loro visualizzazione. Questo capitolo continua quella discussione e introduce molte delle funzioni disponibili in QBasic per lavorare con le stringhe. In particolare impareremo:

- Come assegnare l'input dell'utente a una stringa.
- Come combinare delle stringhe.
- Come selezionare dei caratteri da una stringa.
- Come paragonare delle stringhe.
- Come ordinare delle stringhe.

Insieme alle tecniche che avete appreso nei capitoli precedenti, queste funzioni vi saranno utili per eseguire una grande varietà di compiti.

DESCRIZIONE DELLE STRINGHE

Una stringa è una serie di caratteri consecutivi che si usano come una unità. Di solito le informazioni vengono memorizzate come stringhe quando non si riesce a memorizzarle facilmente come dati di tipo numerico. Per esempio il titolo di un libro, diciamo *Capitanti coraggiosi*, per la sua natura testuale viene memorizzato come stringa piuttosto che come dato di tipo numerico.

Può essere comodo pensare che una stringa occupi una serie di posizioni di memoria nel computer e che queste siano come delle piccole caselle poste una accanto all'altra, con un carattere ciascuna al loro interno (figura 9-1). Le posizioni di memoria (caselle) sono fisse ma i caratteri possono spostarsi liberamente da una casella all'altra ed anche aggiunti o tolti a seconda delle esigenze.

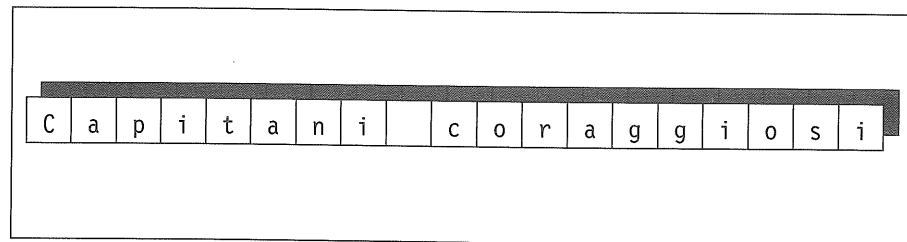



Figura 9-1. *Una stringa è una sequenza di caratteri.*

In una stringa si possono usare i seguenti caratteri:

- Lettere maiuscole dell'alfabeto (dalla A alla Z).
- Lettere minuscole dell'alfabeto (dalla a alla z).
- Numeri (da 0 a 9).
- Simboli di punteggiatura (.,;:"?!).
- Simboli matematici (%()-+=\/<>).
- Simboli eterogenei (~ $ ^ & * _ { } []).
- Caratteri del set di caratteri estesi IBM.

DUE TIPI DI STRINGHE

QBasic permette di usare due tipi di stringhe nei programmi:

- Le *costanti stringa* sono dichiarate all'interno del programma e, come suggerisce il nome, i caratteri in esse contenuti non cambiano mai.
- Le *variabili stringa* sono anch'esse dichiarate all'interno del programma ma i caratteri possono cambiare in qualunque momento.

Esaminiamo ora come e perché si usano questi tipi di stringhe in QBasic.

Costanti stringa

Le costanti stringa sono stringhe che non cambiano mentre si esegue il programma. QBasic separa le costanti stringa in due gruppi: *costanti stringa letterali* e *costanti stringa simboliche*.

Costanti stringa letterali

Una stringa letterale è una serie di caratteri consecutivi circondati da doppie virgolette. Di solito si assegnano le stringhe letterali a una variabile o si usano come argomenti per un enunciato o una funzione. Per esempio, gli enunciati che seguono contengono delle stringhe letterali.

```
evviva$ = "Forza Milan!"

PRINT "Viale dei Tigli, 1313"
```

```
meta$ = StringaCentrale$ ("Oggi farò volare un aquilone")
DatadiNascita$ = 19-11-63"
```

Le stringhe letterali si chiamano anche *valori stringa*.

Costanti stringa simboliche

Una costante stringa simbolica è un nome assegnato a una stringa letterale:

```
CONST DI$ = "Sua Altezza Reale, Diana, Principessa del Galles"
```


Una volta assegnata, una costante può essere usata in tutti i posti in cui si dovrebbe usare una stringa letterale (allo stesso modo in cui si può chiamare qualcuno con un soprannome o col suo nome completo).

Come si è detto nel capitolo 4, una costante viene assegnata in un enunciato CONST, di solito all'inizio del programma. In questo libro le costanti sono scritte tutte in lettere maiuscole per distinguerle dalle variabili:

```
CONST DI$ = "Sua Altezza Reale, Diana, Principessa del Galles"
PRINT "La moglie di Carlo ora si chiama"; DI$
```

Una volta dichiarata una costante, non si può cambiarne il valore. Per esempio, un programma che contiene i due enunciati che seguono finirebbe col generare il messaggio di errore *Duplicate definition* (definizione duplicata) al momento dell'esecuzione:

```
CONST DI$ = "Sua Altezza Reale, Diana, Principessa del Galles"
DI$ = "Lady Diana Spencer"
```

La differenza tra una costante e una variabile è infatti questa: il valore di una variabile può cambiare durante l'esecuzione del programma, quello di una costante invece resta immutato.

Esercizio: Uso delle costanti stringa

Le costanti sono particolarmente utili quando nel programma vi sono motivi ricorrenti. Il programma DITO.BAS (figura 9-2) dichiara la costante DITO$ e vi assegna la stringa letterale *Questo dito dice*.

Scrivete ed eseguite il programma DITO.BAS.

```
' DITO.BAS
' Questo programma mostra l'uso delle costanti stringa.
CLS
CONST DITO$ = "Questo dito dice"
PRINT DITO$; "non c'è pane"
PRINT DITO$; "come faremo"
PRINT DITO$; "lo compreremo"
PRINT DITO$; "ce n'è un pezzettino"
PRINT DITO$; "datelo a me che sono il più piccolino"
```

Figura 9-2. *Una semplice dimostrazione dell'uso di costanti stringa simboliche e letterali.*

Otterrete il seguente risultato:

```
Questo dito dice non c'è pane
Questo dito dice come faremo
Questo dito dice lo compreremo
Questo dito dice ce n'è un pezzettino
Questo dito dice datelo a me che sono il più piccolino
```

Variabili stringa

A differenza delle costanti stringa, le variabili stringa possono cambiare in ogni momento durante l'esecuzione di un programma. QBasic supporta due tipi di variabili stringa: *stringhe a lunghezza variabile* e *stringhe a lunghezza fissa*.

Stringhe a lunghezza variabile

Le stringhe a lunghezza variabile permettono di ottenere e memorizzare delle informazioni fornite dall'utente di un programma. Una stringa a lunghezza variabile può contenere da 0 a 32767 caratteri (sebbene l'intervallo tipico sia da 0 a 80 a causa della larghezza dello schermo) e la sua lunghezza può aumentare o diminuire durante l'esecuzione del programma. Si può dichiarare una stringa a lunghezza variabile in tre modi:

- Mettendo il carattere di dichiarazione del tipo stringa ($) alla fine del nome della variabile. Per esempio, l'enunciato che segue richiede una stringa all'utente e la assegna alla variabile stringa Nome$.

  ```
  INPUT "Scrivi il tuo nome: ", Nome$
  ```

  Esaminando gli esempi in questo libro, si incontreranno molti casi in cui gli enunciati INPUT usano stringhe a lunghezza variabile per assegnare nomi significativi a particolari parole o frasi; queste stringhe si usano anche come argomenti per enunciati e funzioni (come PRINT).

- Usando l'enunciato DEFSTR. Nell'esempio seguente l'enunciato DEFSTR indica a QBasic di considerare come stringa a lunghezza variabile ogni variabile che cominci con la lettera *S* (si può usare anche una lettera diversa) e che non abbia un carattere di dichiarazione del tipo (%,&,! o $):

  ```
  DEFSTR S
  ```

- Usando AS STRING con l'enunciato DIM. L'enunciato che segue dichiara che la variabile *nomeCognome* è una stringa a lunghezza variabile:

  ```
  Cognome AS STRING
  ```

  Si noti che *nomeCognome* non termina con $; infatti, quando si dichiara una stringa usando la parola chiave AS STRING non si può usare il carattere di dichiarazione del tipo.
  Come si è visto nel capitolo precedente, si può anche usare l'enunciato DIM per dichiarare una *matrice* di stringhe a lunghezza variabile. Per esempio l'enunciato che segue dichiara una matrice di 10 stringhe a lunghezza variabile (supponendo che l'enunciato OPTION BASE 1 appaia per primo):

  ```
  $(10)
  ```

Esercizio: Uso di stringhe a lunghezza variabile

Il programma TELEF.BAS (figura 9-3) usa gli enunciati INPUT e una matrice stringa a lunghezza variabile bidimensionale chiamata *contatti$* per memorizzare una lista di amici e i rispettivi numeri di telefono. Poiché, in alcune nazioni, i numeri di telefono possono contenere dei caratteri diversi dai numeri, come linee e lettere, è meglio memorizzarli come stringhe.

Scrivete ed eseguite il programma TELEF.BAS.


```
' TELEF.BAS
' Questo programma usa una matrice stringa a lunghezza
' variabile per registrare nomi e numeri di telefono.
OPTION BASE 1      ' Imposta il limite inferiore della matrice 1
CLS
INPUT "Quanti nomi desiderate scrivere? ", nomi%
PRINT
DIM contatti$(nomi%, 2)  ' Dichiara la matrice per i nomi e i numeri di
telefono
FOR i% = 1 TO nomi%       ' Legge i nomi nella matrice contatti$
        INPUT "Scrivete un nome: ", contatti$(i%, 1)
        INPUT "Scrivete il numero di telefono: ", contatti$(i%, 2)
        PRINT
NEXT i%
PRINT "Avete creato la seguente lista di contatti:"
PRINT
FOR i% = 1 TO nomi%       ' Stampa i contenuti della matrice
        PRINT "Nome: "; contatti$(i%, 1), "Telefono: "; contatti$(i%, 2)
NEXT i%
```

Figura 9-3. *Una dimostrazione di stringhe a lunghezza variabile.*

Al momento dell'esecuzione dovreste ottenere un risultato simile a questo:

```
Quanti nomi desiderate scrivere? 3

Scrivete un nome: Aldo Rossi
Scrivete il numero di telefono: 555-1

Scrivete un nome: Jack Angus
Scrivete il numero di telefono: 555-PLUM

Scrivete un nome: Paul Thorn
Scrivete il numero di telefono: 555-3

Avete creato la seguente lista di contatti:

Nome: Aldo Rossi Telefono: 555-1
Nome: Jack Angus Telefono: 555-PLUM
Nome: Paul Thorn Telefono: 555-3
```

Notate che si ottiene l'allineamento della colonna dei numeri di telefono usando una virgola nell'enunciato PRINT per formare due colonne; in questo caso, il solo avanzare fino alla successiva area di stampa è sufficiente per far allineare le voci. Tuttavia, se i valori della stringa sono di dimensioni molto diverse bisogna trovare un'altra soluzione. Provate a scrivere dei nomi di diverse lunghezze per capire il perché.

Stringhe a lunghezza fissa

Le stringhe a lunghezza fissa permettono di dichiarare una variabile stringa di una certa lunghezza. Sebbene i contenuti di una queste stringhe possano cambiare in qualsiasi momento, la lunghezza rimane costante. In generale, è meglio usare le stringhe a lunghezza fissa quando si è sicuri della lunghezza di una stringa di caratteri o quando bisogna allineare dei gruppi di stringhe.

Per dichiarare una stringa a lunghezza fissa si aggiunge all'enunciato DIM il nome della variabile stringa e la lunghezza, come si vede in questo esempio:

```
DIM nomeStringa AS STRING * n
```

nomeStringa è il nome della stringa a lunghezza fissa; *n* è la lunghezza della stringa. Per esempio, l'enunciato che segue dichiara una stringa a lunghezza fissa (chiamata *indirizzo*) lunga 25 caratteri:

```
DIM indirizzo AS STRING * 25
```

Si può anche dimensionare una *matrice* di stringhe a lunghezza fissa includendo il numero degli elementi e la sua dimensione. Per esempio, l'enunciato che segue dichiara una matrice stringa a lunghezza fissa monodimensionale chiamata *nomeCognome*:

```
DIM nomeCognome(25) AS STRING * 25
```

Le stringhe a lunghezza fissa sono giustificate a sinistra per definizione e sono seguite da spazi vuoti a destra se la lunghezza del valore della stringa assegnato alla stringa stessa è minore della lunghezza totale della stringa (figura 9-4). Si noti che non si può usare il carattere di dichiarazione del tipo stringa ($) quando si dichiara una stringa a lunghezza fissa.

Esercizio: Stringhe a lunghezza fissa

Il programma TELEFOK.BAS (figura 9-5) corregge il programma TELEF.BAS per includere due matrici stringa a lunghezza fissa monodimensio-

Figura 9-4. *Allineamento di caratteri in una stringa a lunghezza fissa.*

nale per creare un elenco telefonico. Sono necessarie due matrici separate perché la lunghezza della stringa in ogni matrice è diversa; se la lunghezza fosse la stessa si potrebbe usare una matrice stringa a lunghezza fissa bidimensionale.

Scrivete ed eseguite il programma TELEFOK.BAS.

```
' TELEFOK.BAS
' Questo programma usa due matrici stringa a lunghezza
' fissa per registrare nomi e numeri di telefono.
OPTION BASE 1      ' imposta la base della matrice a 1
CLS
INPUT "Quanti nomi desiderate inserire? ", nomi%
PRINT
DIM nomeCognome(nomi%) AS STRING * 18 ' 18 caratteri per ciascun nome
DIM telefono(nomi%) AS STRING * 14    ' 14 caratteri per ciascun telefono
FOR i% = 1 TO nomi%       ' legge i valori in entrambe le matrici
      INPUT "Scrivete un nome: ", nomeCognome(i%)
      INPUT "Scrivete il numero di telefono: ", telefono(i%)
NEXT i%
PRINT "Nome Telefono"
PRINT "---------------------------------"
FOR i% = 1 TO nomi%       ' stampa i contenuti di entrambe le matrici
      PRINT nomeCognome(i%); " "; telefono(i%)
NEXT i%
```

Figura 9-5. *Una dimostrazione dell'uso di stringhe a lunghezza fissa.*

Otterrete un risultato simile a questo:

```
Quanti nomi desiderate scrivere? 3

Scrivete un nome: Aldo Rossi
Scrivete il numero di telefono: 555-1

Scrivete un nome: Jack Angus
Scrivete il numero di telefono: 555-PLUM

Scrivete un nome: Paul Thorn
Scrivete il numero di telefono: 555-3

Nome  Telefono
-----------------------------------------
Aldo Rossi  555-1
Jack Angus  555-PLUM
Paul Thorn  555-3
```

Notate che ora si può predire l'allineamento della colonna del numero di telefono. Se uno dei nomi o dei numeri di telefono fosse più lungo del numero assegnato di caratteri, verrebbe troncato.

Combinazione di stringhe

Una delle cose più semplici che si possano fare con le stringhe è quella di combinarle fra loro per formare stringhe più lunghe. Questo processo viene chiamato *concatenazione;* le stringhe vengono concatenate principalmente per preparare il testo come output per lo schermo o la stampante oppure per predisporre le stringhe all'archiviazione in strutture di dati o in un file. Una stringa concatenata di solito contiene al massimo 80 caratteri il che impedisce alla stringa di andare a capo.

Si possono concatenare sia le variabili stringa che le costanti in qualsiasi combinazione e poi assegnarne il risultato a una variabile stringa o usarlo come argomento di un enunciato o di una funzione che richiede valori stringa (come PRINT).

Per esempio, l'enunciato che segue usa l'operatore di concatenazione (+) per combinare le stringhe letterali *Programmare*, *è*, *divertente* e assegna il risultato alla variabile di stringa *frase$*:

```
frase$ = "Programmare" + "è" + "divertente"
```

L'operatore di concatenazione combina le tre stringhe letterali per formare un'unica stringa; l'operatore di assegnazione (=) assegna poi il risultato alla variabile stringa *frase$*. Si noti che in questo enunciato non ci sono spazi nelle stringhe letterali (gli spazi che circondano gli operatori non contano); perciò se si dovesse stampare il valore di *frase$*, usando l'enunciato PRINT, apparirebbe il seguente risultato:

```
Programmareèdivertente
```

L'operatore di concatenazione combina le stringhe esattamente come sono, senza aggiungere spazi. Per includere gli spazi, bisogna aggiungerli alle stringhe letterali, inserendo uno spazio all'interno delle virgolette. Tutti i seguenti enunciati mostrano anche gli spazi:

- `PRINT "Programmare" + "è" + "divertente"`
- `PRINT "Programmare " + "è " + "divertente"`
- `PRINT "Programmare" + " " + "è" + " " + "divertente"`

Si può anche usare direttamente il risultato di una concatenazione come argomento di un enunciato, senza assegnare il risultato a una variabile intermedia come *frase$:*

```
PRINT "Programmare " + "è " + "divertente"
```

Il risultato dell'enunciato precedente è lo stesso risultato di

```
frase$ = "Programmare " + "è " + "divertente"
PRINT frase$
```

Esercizio: Concatenazione di stringhe

Il programma NOTIZIE.BAS (figura 9-6) mostra l'uso di alcune possibilità che vi si offrono attraverso la concatenazione di stringhe.

Scrivete ed eseguite il programma NOTIZIE.BAS.

```
' NOTIZIE.BAS
' Questo programma mostra l'uso della concatenazione di
' stringhe.

CONST STRUTTURA$ = "Il ponte di"

DIM azione AS STRING * 10
azione = "sta cadendo"

direzione$ = "giù"

CLS

INPUT "Scrivi il nome di una città: ", nomeCittà$

PRINT
PRINT "Notizia-lampo: ";
PRINT nomeCittà$ + " " + STRUTTURA$ + " " + azione + " " +
direzione$ + "!"
```

Figura 9-6. *Un programma che mostra l'uso della concatenazione di stringhe.*

Se quando vi si chiede il nome della città rispondete *Londra*, avrete il seguente risultato:

```
Scrivete il nome di una città: Londra
Notizia-lampo: Il ponte di Londra sta cadendo!
```


USO DELLE FUNZIONI STRINGA

Finora si è visto come dichiarare le costanti e le variabili stringa, come combinare le stringhe attraverso la concatenazione e come usare le stringhe come argomenti negli enunciati INPUT e PRINT. In questo paragrafo si parlerà delle funzioni di QBasic specificamente disegnate per manipolare e restituire valori da stringhe letterali a variabili stringa. In particolare vedremo:

- Come cambiare maiuscole o minuscole in una stringa.
- Come determinare la lunghezza di una stringa.
- Come separare delle stringhe.

Cambiare maiuscole e minuscole in una stringa

È facile cambiare le lettere di una stringa da maiuscole in minuscole o viceversa: basta usare le funzioni UCASE$ o LCASE$. Queste funzioni sono utili se si vuole che tutti i dati nel programma siano uniformi e non si sa con certezza se l'utente scriverà il testo in maiuscolo o minuscolo. Si capirà meglio l'importanza di queste informazioni quando, alla fine di questo capitolo, si vedrà come si confrontano le stringhe.

Uso della funzione UCASE$

Per rendere maiuscole tutte le lettere di una stringa si usa la funzione UCASE$:

UCASE$ *(espressionestringa)*

espressionestringa può essere un qualsiasi tipo di stringa. Il valore restituito da UCASE$ può essere assegnato a una variabile stringa o usato come argomento per un enunciato o funzione che accetta valori stringa. UCASE$ ha effetto solo sulle lettere minuscole di *espressionestringa.*

Esercizio: Uso della funzione UCASE$

Il programma MAIUSC.BAS (figura 9-7) mostra come lavora la funzione UCASE$. MAIUSC.BAS dichiara una costante stringa e due variabili stringa e poi usa la funzione UCASE$ per visualizzarle in maiuscolo. Si noti che la funzione UCASE$ ha effetto solo sul risultato dell'enunciato PRINT e dunque non cambia i contenuti della costante SCRITTORE$ o della variabile *indirizzo$* (ciò è bene, perché, come si è già visto, assegnare un nuovo valore a

una costante sarebbe un errore). Quando le stringhe vengono nuovamente visualizzate alla fine del programma, solo la variabile *area$* mantiene le lettere maiuscole, perché così le erano state assegnate in origine.

Scrivete ed eseguite il programma MAIUSC.BAS.

```
' MAIUSC.BAS
' Questo programma mostra l'uso della funzione UCASE$.
CONST SCRITTORE$ = "Sir Arthur Conan Doyle"
indirizzo$ = "1326 Serpentine Avenue"
area$ = UCASE$ ("St. John's Wood")

CLS

PRINT UCASE$ (SCRITTORE$)
PRINT UCASE$ (indirizzo$) + ", " + area$
PRINT
PRINT SCRITTORE$
PRINT indirizzo$ + ", " + area$
```

Figura 9-7. *Una dimostrazione della funzione UCASE$.*

Avrete un risultato simile a questo:

```
SIR ARTHUR CONAN DOYLE
1326 SERPENTINE AVENUE, ST. JOHN'S WOOD

Sir Arthur Conan Doyle
1326 Serpentine Avenue, ST. JOHN'S WOOD
```

Uso della funzione LCASE$

Per rendere minuscole tutte le lettere di una stringa, si usa la funzione LCASE$:

`LCASE$ (espressionestringa)`

espressionestringa può essere un qualsiasi tipo di stringa. Il valore restituito da LCASE$ può essere assegnato a una variabile stringa o usato come argomento per un enunciato o funzione che accetta valori stringa. LCASE$ ha effetto solo sulle lettere maiuscole di *espressionestringa*.

Esercizio: Uso della funzione LCASE$

Il programma MINUSC.BAS (figura 9-8) mostra come lavora la funzione LCASE$. MINUSC.BAS converte le lettere di una stringa da maiuscole a minuscole e assegna la stringa risultante a una variabile e all'enunciato PRINT.

Scrivete ed eseguite il programma MINUSC.BAS.

```
' MINUSC.BAS
' Questo programma mostra l'uso delle funzioni UCASE$ e LCASE$.

CLS

personaggio$ = "Sherlock Holmes"
PRINT personaggio$

personaggio$ = UCASE$ (personaggio$)
PRINT personaggio$

personaggio$ = LCASE$ (personaggio$)
PRINT personaggio$
```

Figura 9-8. *Un programma che dimostra l'uso della funzione LCASE$.*

Avrete il seguente risultato:

```
Sherlock Holmes
SHERLOCK HOLMES
sherlock holmes
```

Come determinare la lunghezza di una stringa

Vi capiterà spesso di voler sapere quanti caratteri vi sono in una stringa; saperlo può essere utile soprattutto per le stringhe a lunghezza variabile, specialmente per quelle immesse dalla tastiera.

Per determinare quanti caratteri (inclusi gli spazi) vi siano in una stringa si usa la funzione LEN (*length*):

`LEN (espressionestringa)`

espressionestringa può essere, ancora una volta, un qualsiasi tipo di stringa. Il valore restituito da LEN può essere assegnato a una variabile intera o usato come argomento per un enunciato o funzione che accetta valori interi.

La seguente routine mostra come la funzione LEN determini il numero di caratteri in una stringa e assegni tale numero a una variabile intera:

```
nomeCognome$ = "Nonna Papera"
lunghezzaNome% = LEN (nomeCognome$)
PRINT nomeCognome$; " è lungo"; lunghezzaNome%; "caratteri"
```

Quando si esegue questa routine, si ha il seguente risultato:

```
Nonna Papera è lungo 12 caratteri
```

Esercizio: Uso della funzione LEN

Il programma MENU.BAS (figura 9-9) mostra come il valore restituito dalla funzione LEN può essere usato come argomento per un enunciato PRINT. Il programma MENU.BAS dichiara tre stringhe a lunghezza variabile che contengono potenziali scelte di portate per l'utente; queste stringhe vengono visualizzate con gli enunciati PRINT.
Un enunciato INPUT richiede all'utente di scegliere una portata; questa selezione viene poi assegnata alla variabile *scelta$* e viene convertita in maiuscole con la funzione UCASE$. L'enunciato SELECT CASE paragona la variabile *scelta$* alle tre variabili di portata; se c'è una corrispondenza, la scelta della portata e la sua lunghezza (restituita dalla funzione LEN) vengono stampate, altrimenti l'enunciato CASE ELSE stampa il messaggio *Non riconosco questa scelta!*

Scrivete ed eseguite il programma MENU.BAS.


```
' MENU.BAS
' Questo programma mostra l'uso della funzione LEN.

sigGrossi$ = "Cappuccino e cornetto"  ' dichiara le stringhe di portata
Simone$ = "Pasta del giorno"
piattoPietro$ = "Frittata di spinaci"

CLS

PRINT "Che cosa desideri?"       ' visualizza suggerimento portate
PRINT

PRINT sigGrossi$    ' Visualizza scelte delle portate
PRINT Simone$
PRINT piattoPietro$
PRINT

INPUT "Scelta: ", scelta$' riceve la scelta dell'utente
scelta$ = UCASE$ (scelta$)       ' Converte la stringa in maiuscolo
PRINT

SELECT CASE scelta$        ' trova la variabile che corrisponde
       CASE IS = UCASE$ (sigGrossi$)   ' alla portata scritta e stampa
                                       ' lunghezza
              PRINT sigGrossi$; " è lungo", LEN(sigGrossi$); "caratteri"
       CASE IS = UCASE (Simone$)
              PRINT Simone$; " ™ lungo"; LEN(Simone$); "caratteri"
```

Figura 9-9. *Un programma che mostra l'uso della funzione LEN.*

Otterrete un risultato simile a questo:

```
Che cosa desideri?

Cappuccino e cornetto
Pasta del giorno
Frittata di spinaci

Scelta: Cappuccino e cornetto

Cappuccino e cornetto è lungo 21 caratteri
```

Come separare delle stringhe

Abbiamo visto come QBasic permetta di combinare delle stringhe attraverso la concatenazione. Talvolta, però, può essere necessario separare delle stringhe, per ottenere, per esempio, solo il cognome di una persona dal nome e cognome. QBasic contiene sei funzioni che permettono di lavorare con parti di stringhe. I paragrafi che seguono descrivono come usare queste funzioni con i seguenti obiettivi:

- Ottenere l'estremità destra di una stringa (RIGHT$).
- Ottenere l'estremità sinistra di una stringa (LEFT$).
- Ottenere il centro di una stringa (MID$).
- Tagliare l'estremità destra di una stringa (RTRIM$).
- Tagliare l'estremità sinistra di una stringa (LTRIM$).
- Trovare una stringa all'interno di un'altra (INSTR).

Si parlerà anche di enunciati che permettono di

- Ottenere una riga intera di input (LINE INPUT$).
- Stampare caratteri ripetuti (SPACE$, STRING$).

Come ottenere le estremità di una stringa

Le funzioni RIGHT$ e LEFT$ consentono di recuperare uno o più caratteri cominciando da una delle due estremità di una stringa; ciò è utile quando si vuole visualizzare o rimuovere parte di una stringa.

La sintassi della funzione RIGHT$ è la seguente:

`RIGHT$ (espressionestringa, n)`

La sintassi della funzione LEFT$ è la seguente:

`LEFT$ (espressionestringa, n)`

espressionestringa può essere un qualsiasi tipo di stringa e *n* è un valore intero (che va da 0 alla lunghezza della stringa) che indica il numero di caratteri che devono essere restituiti da RIGHT$ o LEFT$. Il valore restituito può essere assegnato a una variabile stringa o usato come argomento per un enunciato o una funzione che accetta valori stringa.

Esercizio: Uso della funzione RIGHT$

Il programma GETRIGHT.BAS (figura 9-10) usa la funzione RIGHT$ per recuperare dei caratteri da una variabile chiamata *alfabeto$*, che contiene le 21 lettere dell'alfabeto. GETRIGHT.BAS estrae il numero di caratteri richiesto e li visualizza.

Scrivete ed eseguite il programma GETRIGHT.BAS.

```
' GETRIGHT.BAS
' Questo programma mostra l'uso della funzione RIGHT$.

CLS

alfabeto$ = "ABCDEFGHILMNOPQRSTUVZ"      ' dichiara la stringa
                                         'di testo

PRINT "Quanti caratteri (da destra a sinistra) nella"
PRINT "seguente stringa desideri visualizzare?"
PRINT
PRINT alfabeto$   ' visualizza la stringa di testo
PRINT

        ' riceve dall'utente il numero di caratteri di destra
        'da visualizzare
DO      ' Fa un loop fino a che il numero è nell'intervallo
        'giusto (da 1 a 21)
        INPUT "    Numero (1-21): ", numDestra%
LOOP WHILE (numDestra% < 1) OR (numDestra% > 21)

PRINT
carDestra$ = RIGHT$ (alfabeto$, numDestra%)   ' visualizza
                                              'i caratteri
PRINT "Hai specificato"; LEN (carDestra$); "caratteri: ";
carDestra$
```

Figura 9-10. *Un programma che mostra l'uso della funzione RIGHT$.*

Otterrete un risultato simile a questo:

```
Quanti caratteri (da destra a sinistra) nella
seguente stringa desideri visualizzare?

ABCDEFGHILMNOPQRSTUVZ

   Numero (1-21): 14

Hai specificato 14 caratteri: HILMNOPQRSTUVZ
```

Esercizio: Uso della funzione LEFT$

Il programma GETLEFT.BAS (figura 9-11) corregge il programma GETRIGHT.BAS per estrarre dei caratteri dall'estremità sinistra di una stringa con la funzione LEFT$.

Si noti che i nomi delle variabili sono cambiati di poco (*numDestra%* diventa *numSinistra%*, e *carDestra$* diventa *carSinistra$*) e che la funzione RIGHT$ è stata cambiata in LEFT$. A parte questi cambiamenti (e altri riguardanti il prompt e i commenti al programma), GETLEFT.BAS è identico a GETRIGHT.BAS.

Poiché le operazioni delle funzioni RIGHT$ e LEFT$ sono tanto simili, è facile cambiare il programma per modificare una stringa dall'estremità opposta.

Modificate il programma GETRIGHT.BAS perché vada bene per GETLEFT.BAS ed eseguitelo.

Otterrete un risultato simile al seguente:

```
Quanti caratteri (da sinistra a destra) nella
seguente stringa desideri visualizzare?

ABCDEFGHILMNOPQRSTUVZ

   Numero (1-21): 12

Hai specificato 12 caratteri: ABCDEFGHILMN
```

Come ottenere il centro di una stringa

La funzione MID$ consente di recuperare uno o più caratteri da qualunque parte all'interno di una stringa: da sinistra, dal centro, o (con l'ausilio della


```
' GETLEFT.BAS
' Questo programma mostra l'uso della funzione LEFT$.

CLS

alfabeto$ = "ABCDEFGHILMNOPQRSTUVZ"      ' dichiara la stringa
                                         ' di testo

PRINT "Quanti caratteri (da sinistra a destra) nella
PRINT "seguente stringa desideri visualizzare?"
PRINT
PRINT alfabeto$  ' Visualizza stringa di testo
PRINT

      ' riceve dall'utente il numero di caratteri
      'di sinistra da visualizzare
DO    ' Fa un loop fino a che il numero è nell'intervallo giusto
      '(da 1 a 21)
      INPUT "   Numero (1-21): ", numSinistra%
LOOP WHILE (numSinistra% < 1) OR (numSinistra% > 21)

PRINT
carSinistra$ = LEFT$ (alfabeto$, numSinistra%)      ' Visualizza i
                                                    'caratteri
PRINT "Hai specificato"; LEN (carSinistra$); "caratteri: ";
carSinistra%
```

Figura 9-11. *Un programma che mostra l'uso della funzione LEFT$.*

funzione LEN) da destra. Proprio questa versatilità fa sì che la funzione MID$ sia una delle più utili funzioni stringa e , come si vedrà più avanti, permette di risolvere molti problemi propri dell'uso di stringhe.

La sintassi per la finzione MID$ è la seguente:

MID$ *(espressionestringa, inizio, lunghezza)*

espressionestringa può essere un qualsiasi tipo di stringa, *inizio* è un valore intero tra 1 e la lunghezza di una stringa (che indica il primo carattere che deve essere restituito) e *lunghezza* è un valore intero che indica il numero di caratteri che devono essere restituiti. Il valore restituito da MID$ può esse-

re assegnato a una variabile stringa o usato come argomento per un enunciato o funzione che accetta valori stringa. La figura 9-12 mostra gli elementi della sintassi della funzione MID$.

Gli enunciati che seguono mostrano alcuni usi della funzione MID$. Si notino le grandi possibilità che emergono quando si usa il valore restituito da una funzione come argomento per MID$ o quando si assegna il valore restituito da MID$ a un'altra enunciato o funzione.

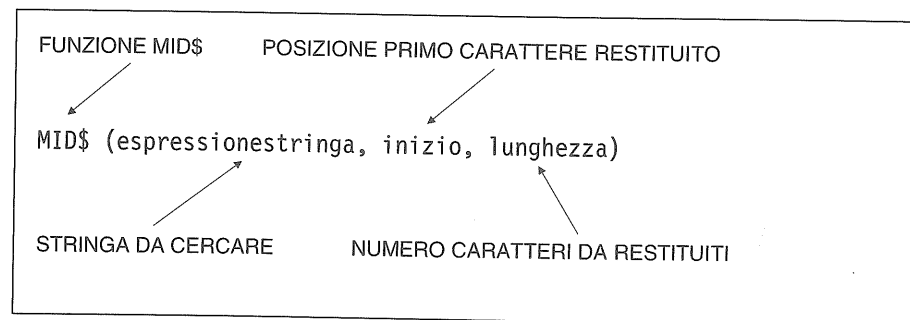



Figura 9-12. *Gli elementi della funzione MID$.*

```
secondoNome$ = MID$ ("Regina Victoria Belfield", 7, 8)
```

Risultato: *secondoNome$* contiene Victoria

```
indirizzo$ = "1521 Plumtree Lane 25-K"
inizioNomeStrada% = 6
lunghezza% = 13
PRINT UCASE$ (MID$(indirizzo$, inizioNomeStrada%, lunghezza%))
```

Risultato: PLUMTREE LANE

```
inString$ = "Oggi è una bella giornata"
parolaPiAùDestra = MID$ (inString$, LEN(inString$) - 4, 5)
```

Risultato: parolaPiùADestra$ contiene giornata

```
PRINT "L'anno corrente  è"; MID$ (DATE$, 7, 4)
```

Risultato: l'anno corrente è 1991

Esercizio: Uso della funzione MID$

Il programma GETMID.BAS (figura 9-13) mostra come recuperare dei caratteri dal mezzo di una stringa con la funzione MID$.

GETMID modifica i programmi GETRIGHT e GETLEFT per includere un punto di partenza insieme al numero di caratteri che devono essere visualizzati nella stringa *alfabeto$*. GETMID usa due loop di tipo DO per ottenere valori interi nell'intervallo ammesso e poi usa la funzione MID$ per assegnare i caratteri selezionati alla variabile *carMezzo$*.

I risultati della selezione vengono stampati con la successiva enunciato IF, che appare quasi alla fine del programma:

```
IF (numDaVisualizzare% = LEN (carMezzo$)) THEN
      PRINT numDaVisualizzare%; "caratteri visualizzati: "; carMezzo$
ELSE
      PRINT numDaVisualizzare%; "caratteri richiesti,";
      PRINT LEN (carMezzo$); "visualizzati: "; carMezzo$
END IF
```

L'enunciato IF confronta *numDaVisualizzare%* (la variabile che contiene il numero di caratteri che l'utente ha richiesto fossero visualizzati) e il numero di caratteri in *carMezzo$* (restituita dalla funzione LEN). Se i due valori sono uguali, il valore di *numDaVisualizzare%* viene stampato insieme al contenuto di *carMezzo$*. Se i due valori non sono uguali, viene usata la funzione LEN per determinare il numero reale di caratteri, e questo valore viene visualizzato insieme alla stringa *carMezzo$*. GETMID contiene questo messaggio aggiuntivo per far sapere all'utente che la lunghezza di visualizzazione digitata eccede il numero di caratteri rimanenti nella stringa.

Modificate il programma GETLEFT.BAS perché vada bene con il programma GETMID.BAS ed eseguitelo.

Al momento dell'esecuzione dovreste ottenere un risultato simile a quello riportato a pagina 243.

```
' GETMID.BAS
' Questo programma mostra l'uso della funzione MID$.

CLS

alfabeto$ = "ABCDEFGHILMNOPQRSTUVZ"   ' dichiara la stringa di testo

PRINT "Quanti caratteri (da sinistra verso destra) nella
PRINT "seguente stringa desideri visualizzare?"
PRINT
PRINT alfabeto$    ' visualizza la stringa di testo
PRINT

        ' riceve dall'utente il numero di caratteri da visualizzare
DO      ' usa un loop fino a che il numero è nell'intervallo
        ' giusto (da 1 a 21)
        INPUT "   Numero (1-21): ", numDaVisualizzare%
LOOP WHILE (numDaVisualizzare% < 1) OR (numDaVisualizzare% > 21)
PRINT ' riceve il numero iniziale...
PRINT "Con quale carattere desideri cominciare?"
PRINT

DO      '...nell'intervallo giusto
        INPUT "   Numero di inizio (1-21): ", inizio%
LOOP WHILE (inizio% < 1) OR (inizio% > 21)
PRINT ' riceve i caratteri
carMezzo$ = (alfabeto$, inizio%, numDaVisualizzare%)

        ' paragona i caratteri richiesti con i caratteri reali recuperati
        ' e stampa un messaggio appropriato con una stringa
IF (numDaVisualizzare% = LEN (carMezzo$)) THEN
        PRINT numDaVisualizzare%; "caratteri visualizzati: "; carMezzo
ELSE
        PRINT numDaVisualizzare%; "caratteri richiesti,";
        PRINT LEN (carMezzo$); "visualizzati: "; carMezzo$
END IF
```

Figura 9-13. *Un programma che mostra l'uso della funzione MID$.*


```
Quanti caratteri (da sinistra verso destra) nella
seguente stringa desideri visualizzare?

ABCDEFGHILMNOPQRSTUVZ

   Numero (1-21): 14
Con quale carattere desideri cominciare?

   Numero di inizio (1-21): 4

14 caratteri visualizzati: DEFGHILMNOPQRS
```

Se specificate un numero fuori dall'intervallo ammesso da GETMID, otterrete un risultato come questo:

```
Quanti caratteri (da sinistra verso destra) nella seguente stringa
desideri visualizzare?

ABCDEFGHILMNOPQRSTUVZ

   Numero (1-21): 0
   Numero (1-21): 30
   Numero (1-21): 15
Con quale carattere desideri cominciare?

   Numero iniziale (1-21): v

Ricomincia da capo
   Numero di inizio (1-21): 18

15 caratteri richiesti, 3 visualizzati: UVZ
```

Se scrivete un valore non numerico (come una lettera) in risposta a uno dei prompt, QBasic stampa il messaggio *Ricominciare da capo* e visualizza di nuovo il prompt. Trattare con questo tipo di risposta da parte dell'utente è importante per sviluppare programmi "a prova di errore".

Come tagliare una stringa

Quando si vogliono allineare stringhe o eliminare spazi vuoti alle loro estremità, si possono usare le funzioni di taglio di QBasic, LTRIM$ e RTRIM$. La funzione LTRIM$ permette di tagliare l'estremità sinistra di una stringa

restituendola senza spazi davanti o tabulazioni. La funzione RTRIM$ svolge una funzione simile, all'estremità destra.

La sintassi per la funzione LTRIM$ è la seguente:

```
LTRIM$ (espressionestringa)
```

La sintassi per la funzione RTRIM$ è la seguente:

```
RTRIM$ (espressionestringa)
```

espressionestringa può essere un qualsiasi tipo di stringa. Il valore restituito da LTRIM$ e RTRIM$ può essere assegnato a una variabile stringa o usato come argomento per un enunciato o funzione che accetta valori stringa.

La routine che segue usa la funzione LTRIM$ per rimuovere i tre spazi che precedono la variabile stringa *fraseInglese$* prima di stamparla:

```
fraseInglese$ = "   Good evening"
fraseInglese$ = LTRIM$ (fraseInglese$)

PRINT fraseInglese$; "!"
```

Questa routine ha come risultato:

```
Good evening!
```

La routine che segue usa la funzione RTRIM$ per rimuovere i tre spazi finali dalla variabile stringa *fraseTedesca$* prima di stamparla:

```
fraseTedesca$ = "Guten Abend   "
fraseTedesca$ = RTRIM$ (fraseTedesca$)

PRINT fraseTedesca$; "!"
```

Questa routine ha come risultato:

```
Guten Abend!
```

Esercizio: Uso di LTRIM$ e RTRIM$

Il programma SALUTI.BAS (figura 9-14) mostra come usare insieme le funzioni LTRIM$ e RTRIM$ per tagliare degli spazi vuoti da entrambe le estremità di una stringa a lunghezza fissa. SALUTI dichiara due variabili stringa a lunghezza fissa chiamate *nome* e *cognome* e usa gli enunciati INPUT per ottenere valori stringa per queste variabili; il programma poi visualizza due messaggi per l'utente: il primo messaggio non è tagliato, il secondo è tagliato dalle funzioni LTRIM$ e RTRIM$. Notate che le stringhe a lunghezza fissa

in se stesse non vengono modificate (non si può infatti cambiare la lunghezza di una stringa a lunghezza fissa).

Notate anche come un argomento stringa viene trasferito da RTRIM$ a LTRIM$ a PRINT in questo enunciato tratto da SALUTI:

```
PRINT "Penso che"; LTRIM$ (RTRIM$(nome)); " ";
```

La funzione RTRIM$ elabora la variabile *nome* e passa il valore stringa risultante all'enunciato PRINT; questo lo combina con due altri valori stringa e li visualizza insieme sullo schermo.

Scrivete ed eseguite il programma SALUTI.BAS.

```
' SALUTI.BAS
' Questo programma mostra l'uso delle funzioni LTRIM$ e RTRIM$.

DIM nome AS STRING * 12   ' Dichiara le stringhe a lunghezza fissa
DIM cognome AS STRING * 15

CLS

        ' Riceve nome e cognome
INPUT "Scrivi il tuo nome: ", nome
INPUT "Scrivi il tuo cognome: ", cognome

PRINT ' stampa saluto senza tagliarlo
PRINT "Piacere, "; nome; " "; cognome; "!"

        ' stampa complimento senza tagliarlo
PRINT "Penso che"; LTRIM$ (RTRIM$(nome)); " ";
PRINT LTRIM$ (RTRIM$(cognome)); "sia un bel nome."
```

Figura 9-14. *Un programma che mostra l'uso delle funzioni LTRIM$ e RTRIM$.*

Otterrete un risultato simile a questo:

```
Scrivi il tuo nome: Gino
Scrivi il tuo cognome: Rossi

Piacere, Gino     Rossi !
Credo che Gino Rossi sia un bel nome.
```

Come ottenere dall'utente un'intera riga di input

In questo capitolo è stata usato l'enunciato INPUT per ottenere informazioni dall'utente. Questa enunciato è molto versatile perché può assegnare l'input a una o più variabili di diverso tipo e fornire un altro prompt per controllare esattamente ciò che l'utente dovrebbe specificare. Una cosa che non abbiamo ancora discusso è come l'enunciato INPUT tratti una virgola nella riga di input. Si consideri il seguente enunciato, che chiede all'utente di scrivere nome e indirizzo:

```
INPUT "Scrivi nome e indirizzo: ", indirizzoPostale$
```

Se l'utente risponde al prompt con una stringa contenente delle virgole, appare il messaggio di errore *Ricomniciare da capo*, come in questo caso:

```
Scrivi nome e cognome: Mario Bianchi, via Roma 40, 35121 Padova
Ricominciare da capo
Scrivi nome e indirizzo:
```

Come si è visto nel capitolo 4, l'enunciato INPUT fa un uso particolare della virgola nella riga di input: essa serve a separare i valori assegnati alle variabili. Ma cosa succede quando appaiono nell'input delle virgole inaspettate, come nell'esempio sopra? In questo caso si potrebbe risolvere il problema assegnando parti della stringa di informazioni a variabili distinte. Per esempio, il seguente enunciato INPUT assegna il valore stringa digitato dall'utente a cinque variabili:

```
INPUT "Scrivi nome e cognome: ", nome$, indirizzo$, città$, stato$,
cap$
```

Ma ci sono casi in cui sarebbe più semplice avere un solo nome di variabile associato a una riga di input. Si può risolvere questo problema con l'enunciato LINE INPUT. L'enunciato LINE INPUT riceve dalla tastiera un'intera riga di testo e la assegna a una variabile stringa, anche se ci sono delle virgole.

La sintassi per l'enunciato LINE INPUT è la seguente:

```
LINE INPUT [;] ["stringaprompt";] variabilestringa
```

stringaprompt è una stringa letterale che richiede all'utente un input, e *variabilestringa* può essere una qualsiasi variabile stringa.

- Mettendo un punto e virgola immediatamente dopo LINE INPUT il cursore rimane sulla stessa riga dopo che l'utente ha premuto Invio.

- Se *stringaprompt* è inclusa nell'enunciato, si deve inserire un punto e virgola per separare *stringaprompt* da *variabilestringa.*
- A differenza dell'enunciato INPUT, l'enunciato LINE INPUT non stampa punti interrogativi a meno che il punto interrogativo non sia incluso in *stringaprompt.*

La seguente routine dimostra l'utilità di LINE INPUT per lunghe righe di informazioni contenenti virgole:

```
LINE INPUT "Scrivi nome e cognome: "; indirizzoPostale$
PRINT indirizzoPostale$
```

Quando si esegue questa routine e si scrivono le informazioni che si cercava di scrivere prima, si otterrà il seguente risultato:

```
Scrivi nome e cognome: Mario Bianchi, via Roma 40, 35121 Padova
Mario Bianchi, via Roma 40, 35121 Padova
```

Troveremo ancora l'enunciato LINE INPUT negli ultimi capitoli.

Come stampare caratteri ripetuti

In QBasic vi sono due utili funzioni che generano stringhe di caratteri ripetuti: la funzione SPACE$, che restituisce una stringa di spazi, e la funzione STRING$, che restituisce una stringa di caratteri. Entrambe le funzioni aiutano a costruire velocemente stringhe da usare nella formattazione e nell'allineamento del risultato del programma.

La sintassi della funzione SPACE$ è la seguente:

```
SPACE$ (n)
```

n è un valore intero che specifica il numero di spazi che conterrà la stringa. Il valore restituito da SPACE$ può essere assegnato a una variabile stringa o usato come argomento di un enunciato o funzione che accetta valori stringa.

L'uso più comune della funzione SPACE$ è quello della formattazione del risultato, come si può vedere nella routine che segue:

```
vuoto$ = SPACE$ (15)
PRINT vuoto$; "Grande svendita oggi!"
```

Si può usare SPACE$ ogni volta che si rientra costantemente il testo con un numero definito di spazi.

La sintassi della funzione STRING$ è la seguente:

```
STRING$ (m, espressionestringa)
```

m è un valore intero che specifica la lunghezza della stringa che deve essere restituita; *espressionestringa* è il carattere che dev'essere ripetuto. Il valore restituito da STRING$ può essere assegnato a una variabile stringa o usato come argomento per un enunciato o funzione che accetta valori stringa.

Esercizio: Uso di SPACE$ e STRING$

La funzione STRING$ è usata soprattutto per i titoli nel risultato del programma. Il programma INTEST.BAS (figura 9-15) usa STRING$ per visualizzare un messaggio di intestazione nel mezzo dello schermo. Si noti che usare una costante per specificare il numero di caratteri ripetuti semplifica la modifica del programma e che le variabili create da SPACE$ e STRING$ sono molto adatte per la concatenazione.

Scrivete ed eseguite il programma INTEST.BAS.

```
' INTEST.BAS
' Questo programma mostra l'uso delle funzioni SPACE$ e STRING$.

CONST LENGTH% = 16

nomeFile$ = "FAGIOLO.DOC"
vuoto$ = SPACE$ (LENGTH%)
asterisco$ = STRING$ (LENGTH%, "*")
titolo$ = vuoto$ + asterisco$ + " " + nomeFile$ + " " +
asterisco$
CLS

PRINT titolo$
```

Figura 9-15. *Un programma che mostra l'uso delle funzioni SPACE$ e STRING$.*

Il nome del file FAGIOLO.DOC apparirà nel mezzo della riga superiore dello schermo, circondato da un numero uguale di asterischi e spazi vuoti.

Come trovare una stringa all'interno di un'altra stringa

Le funzioni RIGHT$, LEFT$ e MID$ sono state usate per restituire caratteri dalle parti destre, sinistre e centrali di una stringa. Queste funzioni sono efficaci quando si devono estrarre dei caratteri da una specifica posizione nella stringa, ma sono meno efficaci quando devono ricercare ed estrarre un specifico campione. La funzione INSTR cerca una stringa all'interno di un'altra e insieme a RIGHT$, LEFT$ e MID$ e alle funzioni di supporto di cui si è parlato in questa sezione (UCASE$, LCASE$, LEN$, RTRIM$,LTRIM$, SPACE$ e STRING$) completa il quadro degli strumenti utili per lavorare con le stringhe.

La sintassi della funzione INSTR è la seguente:

```
INSTR ([inizio, ] stringabase, cercastringa)
```

inizio è un valore intero facoltativo che specifica il carattere dal quale cominciare la ricerca, *stringabase* è la stringa all'interno della quale viene fatta la ricerca e *cercastringa* è la stringa cercata. Il valore restituito da INSTR può essere assegnato a una variabile intera o usato come argomento per un enunciato o funzione che accetta valori interi. La tavola che segue dà una lista dei valori che la funzione INSTR può restituire:

Condizione	Valore intero restituito
cercastringa trovata in *stringabase*	Posizione in *stringabase* in cui è stato trovato l'abbinamento
cercastringa non trovata in *stringabase*	0
inizio è più grande della lunghezza di *stringabase*	0
stringabase non contiene caratteri	0
cercastringa non contiene caratteri	*inizio* (se esiste); altrimenti 1

Esercizio: Uso della funzione INSTR

Il programma GATTA.BAS (figura 9-16) usa la funzione INSTR per cercare la stringa *atta* nella stringa *C'era una gatta matta matta.*

Scrivete ed eseguite il programma GATTA.BAS.

```
' GATTA.BAS
' Questo programma mostra l'uso della funzione INSTR.

CLS

cerca$ = "atta"
base$ = "C'era una gatta matta matta"
posizStringa% = INSTR(1, base$, cerca$)

IF (posizStringa% <> 0) THEN
     PRINT cerca$; "appare per la prima volta a cominciare dal
carattere"; posizStringa%
ELSE
     PRINT cerca$; "non trovata"
END IF
```

Figura 9-16. *Un programma che mostra l'uso della funzione INSTR.*

L'output del programma sarà:

```
atta appare per la prima volta a cominciare dal carattere 11
```

Sebbene il campione *atta* appaia tre volte in *base$*, la funzione INSTR restituisce soltanto la posizione della prima volta in cui si presenta. Si può tuttavia usare INSTR all'interno di un loop per trovare tutte le ripetizioni di tale campione.

È bene controllare il valore restituito da INSTR quando si usa tale funzione. Ciò consentirà infatti di decidere cosa fare se la stringa cercata non è stata trovata o se la stringa cercata o la stringa base sono vuote.

Per esempio, nel programma GATTA, se non si fosse trovata *atta*, la funzione INSTR avrebbe assegnato il valore 0 alla variabile *posizStringa%* e l'enunciato IF avrebbe visualizzato il messaggio per indicare che *atta* non c'era in *base$*:

```
atta non trovata
```

Esercizio: Trovare tutte le ripetizioni di un campione

Il programma RIPETI.BAS (figura 9-17) usa la funzione INSTR per trovare tutte le ripetizioni di un campione in una serie di righe digitate sulla tastiera.

La funzione *Ripeti%* rappresenta il centro di RIPETI; tutte le volte che INSTR trova la stringa cercata (*cerca$*) nella stringa base (*base$*), la posizione della stringa cercata è assegnata alla variabile *carCorrente%*. La variabile *num%* viene incrementata e la variabile *carCorrente%* viene spostata subito dopo la stringa corrispondente nella stringa base (nuovo punto di partenza per la ricerca); questo processo continua fino a quando si raggiunge la fine della stringa base o la stringa cercata non viene più trovata. La funzione *Ripeti%* restituisce allora il risultato della ricerca alla variabile *rigaRipeti%* nel programma principale. Scrivete ed eseguite il programma RIPETI.BAS. Notate che la figura 9-17 mostra come appare RIPETI.BAS quando viene stampato, non come appare all'interno di QBasic. Assicuratevi di usare il comando Nuovo SUB nel menu Modifica per inserire il sottoprogramma *RiceviTesto* e il comando Nuova FUNCTION nel menu Modifica per inserire la funzione *Ripeti%*. Potete rivedere questo procedimento nel capitolo 7.


```
' RIPETI.BAS
' Questo programma chiede all'utente di specificare un numero
definito di righe
' e una stringa da cercare e poi stampa le righe e il numero di
' abbinamenti trovati.

' Imposta il numero massimo di righe che possono essere
' indicate e dichiara la matrice stringa per conservarle.

CONST MAXLINES% = 10
DIM righeInput$ (MAXLINES%)

' dichiara il sottoprogramma RiceviTesto e la funzione Ripeti%

DECLARE SUB RiceviTesti (righeInput$ (), numDiRighe%)
DECLARE FUNCTION Ripeti% (cerca$, base$)

CLS

' chiama il sottoprogramma RiceviTesto per ricevere l'input '
dall'utente;   numDiRighe% conterrà il numero di righe ricevuto

RiceviTesto righeInput$ (), numDiRighe%
```

Figura 9-17. *Un programma che scorre le righe in cerca di un campione e stampa il numero degli abbinamenti trovati (continua).*

```
' Riceve dall'utente il campione da cercare

PRINT
INPUT "Scrivi la stringa da cercare: ", campione$
PRINT

' chiama la funzione Ripeti% per determinare il numero
' di abbinamenti per riga; la variabile ripetiTotale%
' tiene nota del numero di abbinamenti totali

FOR i% = 1 TO numDiRighe%
      ripetiRiga% = Ripeti% (campione$, righeInput$ (i%))
      ripetiTotale% = ripetiTotale% + ripetiRiga%
NEXT i%

' visualizza le righe inserite dall'utente...

PRINT "Hai inserito le seguenti righe:"
PRINT
FOR i% = 1 TO numDiRighe%
      PRINT righeInput$ (i%)
NEXT i%

' ...e il numero totale di abbinamenti

PRINT
PRINT "Il campione '"; campione$; "' appare"; ripetiTotale%;
"volte"

END

SUB RiceviTesto (matriceStringa$ (), conto%)

' il sottoprogramma RiceviTesto riempie la matrice
' matriceStringa$
' col testo specificato dalla tastiera. Il numero di righe che
' possono  essere immesse è determinato
' dalla costante globale MAXLINES%.
' Sia matriceStringa$ che conto$ (il numero di righe immesso)
```

Figura 9-17. *Un programma che scorre le righe in cerca di un campione e stampa il numero degli abbinamenti trovati (continua).*


```
PRINT "Scrivi fino a"; MAXLINES%; "righe di testo; per
PRINT "finire, "; premi Invio in una riga nuova."
PRINT
conto% = 0

DO
      LINE INPUT "-> "; inLine$    ' Riceve una riga dall'utente
      IF (inLine$ <> "") THEN  ' se la riga non è vuota la copia
            conto% = conto% + 1 ' nella matrice matriceStringa$
            matriceStringa (conto%) = inLine$
      END IF
      ' entra in un loop fino a che conto% = MAXLINES%
      ' o si riceve una riga vuota
LOOP WHILE (conto% < MAXLINES%) AND (inLine$ <> "")

END SUB

FUNCTION Ripeti% (cerca$, base$)
' la funzione Ripeti% restituisce il numero di volte che
' viene trovata in una stringa base una stringa cercata.

lunghezzaCerca% = LEN (cerca$)     ' determina la lunghezza della
                                   ' stringa cercata
lunghezzaBase% = LEN (base$) ' determina la lunghezza
                             ' della stringa base
carCorrente% = 1 ' scostamento del carattere nella stringa base
num% = 0    ' Numero totale di abbinamenti trovati nella base

' entra in un un loop fino a che non viene elaborata l'intera
' stringa o fino a che INSTR non restituisce uno 0

WHILE (carCorrente% <= lunghezzaBase%) AND (carCorrente% <> 0)
      carCorrente% = INSTR (carCorrente%, base$, cerca$)
      IF (carCorrente% <> 0) THEN ' se non 0, è stato trovato un
                                  ' abbinamento
      num% = num% + 1  ' incrementa il numero degli abbinamenti
      ' il nuovo scostamento è uguale allo scostamento
      ' corrente più la lunghezza della stringa cercata
```

Figura 9-17. *Un programma che scorre le righe in cerca di un campione e stampa il numero degli abbinamenti trovati (continua).*

```
        carCorrente% = carCorrente% + lunghezzaCerca%
        END IF
WEND

Ripeti% = num%    ' restituisce il numero totale degli abbinamenti

END FUNCTION
```

Figura 9-17. *Un programma che scorre le righe in cerca di un campione e stampa il numero degli abbinamenti trovati.*

Al momento dell'esecuzione otterrete un risultato simile a questo:

```
Scrivi fino a 10 righe di testo; per finire, premi Invio in una
riga nuova.

-> Dieci bravi regnanti
-> Nove dame danzanti
-> Otto fanciulle vocianti
-> Sette cigni svolazzanti
-> Sei oche starnazzanti
-> Cinque anelli da mano
-> Quattro uccelli da richiamo
-> Tre pesci all'amo
-> Due tortorelle, e
-> Un merlo nero sopra un pero.

Scrivi la stringa da cercare: anti

Hai scritto le seguenti righe:

-> Dieci bravi regnanti
-> Nove dame danzanti
-> Otto fanciulle vocianti
-> Sette cigni svolazzanti
-> Sei oche starnazzanti
-> Cinque anelli da mano
-> Quattro uccelli da richiamo
-> Tre pesci all'amo
-> Due tortorelle, e
-> Un merlo nero sopra un pero.
Il campione 'anti' appare 5 volte
```


CONFRONTO DI STRINGHE

Nel capitolo 4 si è visto che si possono confrontare due stringhe e usarne una in un altra posizione nel programma in base al risultato del confronto. Il seguente enunciato IF confronta la variabile *risposta$* con la stringa letterale *Y* e stampa un messaggio in base al confronto effettuato:

```
risposta$ = "Y"
IF (risposta$ = "Y") THEN
     PRINT "I due valori stringa sono uguali."
ELSE
     PRINT "I due valori stringa non sono uguali."
END IF
```

Se si eseguono gli enunciati sopra riportati, si avrà il seguente risultato:

```
I due valori stringa sono uguali.
```

Se si cambia il valore di *risposta$* in y (minuscolo) e poi si esegue l'enunciato, si avrà un risultato diverso.

```
I due valori stringa non sono uguali.
```

Perché? Dopo tutto Y è uguale a y ... oppure no? Quali criteri usa QBasic per paragonare le stringhe?

Il set di caratteri ASCII

Prima di confrontare due caratteri, QBasic deve convertire ciascun carattere in un numero usando una tabella di traduzione chiamata *set di caratteri ASCII*. QBasic poi confronta i numeri, detti *codici ASCII*, e restituisce il valore logico vero se i codici ASCII sono uguali o *falso* se i codici non lo sono.

La sigla ASCII

Come molti altri termini usati nel linguaggio dei computer, ASCII è una sigla, che sta per American Standard Code for Information Interchange. La parola chiave in questa sigla è *codice*; infatti, come il codice Morse (usato in radio e in telegrafia), anche l'ASCII è un codice accettato internazionalmente per rappresentare caratteri, usato però nei computer e nelle telecomunicazioni. Nell'appendice B troverete la lista completa del set di caratteri ASCII.

Ogni carattere nel set ASCII è associato a un solo numero; il set contiene in tutto 128 caratteri (codici da 0 a 127):

- Caratteri di controllo (codici da 0 a 31), che includono caratteri che corrispondono a tasti particolari della tastiera come Invio, ed Esc.
- Simboli di punteggiatura, numeri e simboli matematici (codici da 32 a 64).
- Lettere maiuscole dell'alfabeto (codici da 65 a 90).
- Lettere minuscole dell'alfabeto (codici da 97 a 122).
- Simboli eterogenei (codici da 91 a 96 e da 123 a 127).

Il codice ASCII per la lettera maiuscola A è 65; il codice ASCII per la lettera minuscola z è 122. Seguendo questa logica, si capisce ora perché, per QBasic, la lettera maiuscola Y e la minuscola y sono due caratteri diversi.

Il set di caratteri estesi IBM

L'appendice B contiene un'altra raccolta di caratteri (codici da 128 a 255) nota come *set di caratteri esteso IBM*. Questo set di simboli è stato sviluppato dall'IBM per la famiglia di personal computer IBM ed è stato successivamente adottato dalla maggior parte delle industrie di personal computer. Il set di carattere esteso IBM (talvolta chiamato anche set di caratteri ASCII superiore) contiene caratteri di lingue straniere, caratteri per disegnare righe o caselle e simboli matematici. Tali caratteri però non appaiono sulla tastiera; per usarli si deve tenere premuto il tasto Alt e digitare 3 cifre del codice esteso IBM (bisogna usare i tasti del tastierino numerico piuttosto che i tasti numerici che si trovano sulla parte superiore della tastiera). Per esempio, per visualizzare il simbolo matematico π sullo schermo, si deve tener premuto Alt e digitare 227 sul tastierino numerico. Il seguente enunciato PRINT mostra un valido uso del set di caratteri esteso IBM in un programma di QBasic. Il simbolo ¿ è stato scritto tenendo premuto Alt e digitando 168.

```
PRINT "¿ Como se llama?"
```

NOTA: Alcuni vecchi monitor non possono visualizzare i caratteri del set di caratteri esteso IBM e ci potranno essere dei problemi per stampare la maggior parte dei simboli di tale set di caratteri se la stampante non è stata appositamente dotata di strumenti in grado di gestirli. Bisogna essere pronti a trovarsi di fronte a strani risultati (innocui però) quando si usa questo set di caratteri.

Conversione dei codici ASCII in caratteri

Se si conosce un codice ASCII ma non si è sicuri quale carattere rappresenti, si può usare la funzione CHR$ per visualizzare il simbolo in questione sullo schermo o nel programma.

La sintassi per la funzione CHR$ è la seguente:

```
CHR$ (codice)
```

codice è un valore intero che indica un codice ASCII o IBM. Il valore restituito da CHR$ può essere assegnato a una variabile stringa o usato come argomento per un enunciato o funzione che accetta valori stringa.

Esercizio: Uso della funzione CHR$

Il programma ASCII.BAS (figura 9-18) mostra come usare la funzione CHR$ per visualizzare il set esteso di caratteri IBM e il set ASCII. Si noti che il programma salta i primi 32 caratteri (di controllo) del set di caratteri ASCII e fa una pausa dopo ogni multiplo di 23 righe (eccetto il primo e l'ultimo set) così si possono osservare i risultati con calma.

Scrivete ed eseguite il programma ASCII.BAS (per ragioni di spazio, in questo caso non mostreremo il risultato del programma).

```
' ASCII.BAS
' Questo programma visualizza i set
' di carattere esteso IBM e ASCII.

CLS

FOR i% = 33 TO 255
    PRINT "Codice"; i%; "= "; CHR$ (i%)
    IF (i% MOD 23 = 0) THEN INPUT "Premi Invio per continuare...", finto$
NEXT%
```

Figura 9-18. *Un programma che usa la funzione CHR$ per visualizzare i set di caratteri IBM e ASCII.*

Conversione di caratteri in codici ASCII

Oltre alla funzione CHR$ esiste anche la funzione ASC che esegue il compito contrario, converte cioè un carattere nel suo rispettivo codice.

La sintassi della funzione ASC è la seguente:

ASC (*espressionestringa*)

espressionestringa è una stringa composta da un solo carattere. Il valore intero restituito da ASC può essere assegnato a una variabile intera o usato come argomento per un enunciato o funzione che accetta valori interi.

Esercizio: Uso della funzione ASC

Il programma CONFR.BAS (figura 9-19) usa la funzione ASC per mostrare come QBasic confronta i caratteri. CONFR chiede all'utente due caratteri e poi usa l'enunciato SELECT CASE per visualizzare uno dei tre messaggi basati sui valori numerici restituiti da ASC.

Scrivete ed eseguite il programma CONFR.BAS.

```
' CONFR.BAS
' Questo programma fa un confronto tra due caratteri.

CLS

INPUT "Scrivi un carattere qualsiasi: ", primoCar$
INPUT "Scrivi un altro carattere: ", secondoCar$
PRINT

SELECT CASE ASC (primoCar$) Í ASC (secondoCar$)
   CASE IS < 0
      PRINT "'"; primoCar$; "' viene prima '"; secondoCar$; "'"
      PRINT "perché"; ASC (primoCar$); "è minore di"; ASC(secondoCar$)
   CASE IS > 0
      PRINT "'"; primoCar$; "' viene dopo '"; secondoCar$; "'"
      PRINT "perché"; ASC (primoCar$); "è maggiore di"; ASC(secondoCar$)
   CASE ELSE
      PRINT "'"; primoCar$; "' è lo stesso di '"; secondoCar$; "'"
      PRINT "perché"; ASC (primoCar$); "è uguale a"; ASC(secondoCar$)
END SELECT
```

Figura 9-19. *Un programma che usa la funzione ASC per confrontare due caratteri.*

Otterrete un risultato simile a questo:

```
Scrivi un carattere qualsiasi: d
Scrivi un altro carattere: j

'd' viene prima di 'j'
perché 100 è maggiore di 74
```

Uso di operatori relazionali con le stringhe

Oltre alle equivalenze fra caratteri, in QBasic troviamo anche confronti di stringa con i seguenti operatori relazionali:

Operatore	Significato	Operatore	Significato
<>	non uguale	>	maggiore di
=	uguale <	=	minore o uguale a
<	minore di	>=	maggiore o uguale a

Un carattere è "maggiore" di un altro se il suo codice ASCII è maggiore. Per esempio, il valore ASCII della lettera B è maggiore del valore ASCII della lettera A, così l'espressione

```
"A" < "B"
```

è vera, e l'espressione

```
"A" > "B"
```

è falsa.

Quando confronta due stringhe, ognuna delle quali contiene più di un carattere, QBasic comincia a paragonare il primo carattere della prima stringa al primo carattere della seconda e poi va avanti carattere per carattere fino a quando trova una differenza. Per esempio, le stringhe *Mike* e *Michael* sono uguali fino al terzo carattere (K e C). Poiché il valore ASCII di K è maggiore di quello di C, l'espressione

```
"Mike" > "Michael"
```

è vera.

Se QBasic non trova alcuna differenza, le due stringhe sono uguali. Se due stringhe sono uguali per molti caratteri ma una delle stringhe continua e l'altra si ferma, la stringa più lunga è maggiore di quella più corta. Consideriamo questo esempio:

```
"AAAAA" > "AAA"
```

è vera.

Ordinamento di stringhe: il programma STRSORT.BAS

Il programma STRSORT.BAS (figura 9-20) usa l'operatore relazionale <= per confrontare degli elementi di matrice e usa l'enunciato SWAP per spostano gli elementi che sono fuori posto.

STRSORT dichiara una matrice di stringhe detta *lineeInput$* e chiama il sottoprogramma *RiceviTesto*, che è lo stesso sottoprogramma che avete già incontrato in RIPETI.BAS (figura 9-17): *RiceviTesto* legge le righe di testo in una matrice e poi restituisce la matrice (*lineeInput$*) e il numero degli elementi nella matrice (*numDiElementi%*) al programma centrale.

STRSORT chiama poi il sottoprogramma *ShellSort*; si osservino i seguenti enunciati al centro di *ShellSort* che confrontano gli elementi della matrice e poi spostano gli elementi se sono fuori posto:

```
IF matriceStringa$ (j%) <= matriceStringa$ (j% + intervallo%) THEN
EXIT FOR
      ' scambia gli elementi della matrice che sono fuori posto
SWAP matriceStringa$ (j%), matriceStringa$ (j% + intervallo%)
```

L'operatore relazionale <= viene usato per confrontare i due elementi della matrice.

- Se il valore dell'espressione stringa *matriceStringa$ (j%)* è minore del valore dell'espressione stringa *matriceStringa$ (j% + intervallo%)*, gli elementi sono al posto giusto e l'enunciato EXIT FOR fa terminare il loop di tipo FOR.
- Se il valore della seconda espressione stringa è maggiore del valore della prima, gli elementi vengono scambiati grazie all'enunciato SWAP. SWAP, che qui incontrate per la prima volta, cambia i valori di una delle due variabili dello stesso tipo. In questo caso vengono scambiati i valori delle due variabili stringa. Quando la matrice è ordinata, viene trasferita di nuovo al programma principale e stampata per intero da un loop di tipo FOR.

Esercizio: Uso dell'operatore relazionale <= e dell'enunciato SWAP

Scrivete ed eseguite il programma STRSORT.BAS. Se avete scritto il programma RIPETI.BAS (figura 9-17), potete usarlo come base per il programma STRSORT.BAS eliminando la funzione *Ripeti* (premere F2, evidenziare *Ripeti*, e premere Canc); selezionate poi il testo nel programma principale e cancellatelo. Scrivete il sottoprogramma *ShellSort* e il nuovo programma principale e poi salvatelo come STRSORT.BAS.

ShellSort

La logica nel sottoprogramma *ShellSort* è basata sul famoso algoritmo Shell Sort, che ordina una matrice di numeri, pubblicato nel 1959 da Donald Shell. Shell Sort ordina una lista di elementi dividendo ripetutamente la lista principale in sottoliste più piccole della metà e confrontando gli elementi in cima e alla fine delle sottoliste. Se gli elementi sono fuori posto, vengono cambiati. Il risultato finale è una matrice di voci in ordine decrescente.

```
' STRSORT.BAS
' Questo programma richiede all'utente una lista di nomi e
' poi ordina i nomi alfabeticamente.
' Imposta un numero massimo di righe da scrivere e dichiara
' una matrice stringa per conservare le righe

CONST MAXLINES% = 5
DIM lineeInput$ (MAXLINES$)

' dichiara i sottoprogrammi RiceviTesto e ShellSort

DECLARE SUB RiceviTesto (matriceStringa$(), numDiElementi%)
DECLARE SUB ShellSort (matriceStringa$(), numDiElementi%)

CLS

' chiama il sottoprogramma RiceviTesto per ricevere l'input
' dall'utente; la variabile numDiElementi% conterrà
' il numero di righe ricevuto
```

Figura 9-20. *Un programma che ordina un elenco di stringhe digitate dall'utente (continua).*

```
RiceviTesto lineeInput$ (), numDiElementi%

' chiama il sottoprogramma ShellSort per mettere la matrice

' lineeInput$ in ordine alfabetico
ShellSort lineeInput$ (), numDiElementi%

PRINT
PRINT "Risultati dell'ordinamento:"
PRINT

FOR i% = 1 TO numDiElementi%    ' stampa i contenuti
                                'della matrice ordinata
      PRINT lineeInput$ (i%)
NEXT i%

END

SUB RiceviTesto (matriceStringa$ (), conto%)

' il sottoprogramma RiceviTesto riempie  matriceStringa$
' con il testo digitato. Il numero di righe
' che possono essere scritte è
' determinato dalla costante globale MAXLINES%. Sia
' matriceStringa$ che conto% (il numero di righe scritto)
' vengono restituite al programma principale.

PRINT "Scrivi fino a"; MAXLINES%; "righe di testo; per finire, ";
PRINT "; premi Invio in una nuova riga."
PRINT

conto% = 0
DO
      LINE INPUT "->"; inLine$ ' riceve la riga dall'utente
      IF (inLine$ <> "") THEN  ' se la riga non è vuota,
            conto% = conto%  +1      ' la copia in matriceStringa$
            matriceStringa$ (conto%) = inLine$

      END IF

      ' entra in un loop fino a che conto% = MAXLINES% o fino
      ' a che riceve una riga vuota
```

Figura 9-20. *Un programma che ordina un elenco di stringhe digitate dall'utente (continua).*


```
LOOP WHILE (conto% < MAXLINES%) AND (inLine$ <> "")

END SUB

SUB ShellSort (matriceStringa$ (), numDiElementi%)

' il sottoprogramma ShellSort ordina gli elementi di matriceStringa$ e
' restituisce matriceStringa al programma principale. L'argomento
' numDiElementi% contiene il numero degli elementi in matriceStringa$.
' ShellSort ordina gli elementi in ordine decrescente.

intervallo% = numDiElementi% \ 2

DO WHILE intervallo% > 0
   FOR i% = intervallo% TO numDiElementi% - 1
      j% = i% - intervallo% + 1
       FOR j% = (i% - intervallo% + 1) TO 1 STEP - intervallo%
         IF matriceStringa$ (j%) <= matriceStringa$ (j% + intervallo%)
THEN EXIT FOR
        ' scambia gli elementi della matrice che sono fuori posto
         SWAP matriceStringa$ (j%), matriceStringa$ (j% + intervallo%)
      NEXT j%
   NEXT i%

   intervallo% = intervallo% \ 2
LOOP

END SUB
```

Figura 9-20. *Un programma che ordina un elenco di stringhe digitate dall'utente.*

Otterrete un risultato simile a questo:

```
Scrivi fino a 15 righe di testo; per finire, premi Invio in una
nuova riga.

-> Gennari, Michela
-> Triestini, Luigi
-> Gennari, Davide
-> Zucchi, Evelina
-> Gennari, Mario
-> Zucchi, Vittorio
-> Berenga, Riccardo
-> Grandi, Emma
->
```

```
Risultati dell'ordinamento:

Berenga, Riccardo
Gennari, Davide
Gennari, Mario
Gennari, Michela
Grandi, Emma
Triestini Luigi
Zucchi, Evelina
Zucchi, Vittorio
```

SOMMARIO

In questo capitolo si sono affrontate molte nuove funzioni ed enunciati, circa 15 in tutto:

- UCASE$, LCASE$.
- LEN.
- RIGHT$, LEFT$, MID$.
- RTRIM$, LTRIM$.
- LINE INPUT.
- SPACE$, STRING$.
- INSTR.
- CHR$, ASC.
- SWAP.

Si è anche visto come dichiarare, combinare, separare, paragonare e ordinare delle stringhe. Tutti gli strumenti per lavorare con le stringhe visti in questo capitolo sono molto utili perché consentono di lavorare con molti dati (e gran parte di ciò che si fa con i personal computer implica l'uso di grandi quantità di dati). Nel prossimo capitolo si discuterà appunto sui modi più efficaci per gestire grandi quantità di dati.

DOMANDE ED ESERCIZI

1. Vero o Falso: le costanti stringa sono per regola scritte in caratteri maiuscoli.

2. Vero o Falso: la seguente dichiarazione di una stringa a lunghezza fissa è un'enunciato valido:

   ```
   DIM cognome$ AS STRING * 20
   ```

3. Quale risultato produce il seguente enunciato?

   ```
   PRINT "ONE" + "TWO" + "THREE"
   ```

4. Quale dei seguenti valori può essere usato come argomento alla funzione LEN?

   a. Una stringa letterale.
   b. Una costante stringa.
   c. Una variabile stringa.
   d. Il risultato di una funzione stringa.

5. Che cosa è accaduto se quando si chiama la funzione INSTR viene restituito il valore 0?

6. Che cosa visualizza il seguente enunciato?

   ```
   PRINT CHR$ (ASC("h") - 32)
   ```

7. Qual è il codice ASCII per la lettera M?

8. Scrivete un programma che richieda all'utente nome e cognome, converta i nomi in lettere maiuscole e poi li stampi con la formattazione *Cognome, Nome*.

9. Scrivete un programma che riceva dall'utente una stringa a lunghezza variabile, inverta l'ordine dei caratteri nella stringa e visualizzi la nuova stringa.

10. Scrivete un programma che riceva dall'utente un nome composto da tre parti (nome, secondo nome, cognome) con una stringa a lunghezza variabile e copi ciascun nome della stringa in una variabile stringa separata.
    Il programma dovrebbe poter trattare nomi di qualsiasi lunghezza, purché l'utente separi i tre nomi con gli spazi. Il risultato del programma dovrebbe assomigliare a questo:

```
Scrivi una stringa: Queen Victoria Belfield
Nome: Queen
Secondo nome: Victoria
Cognome: Belfield
```


CAPITOLO 10

Lavorare con i file

Nei due precedenti capitoli abbiamo visto come gestire grandi quantità di dati in un programma. In questo capitolo vedremo come salvare i dati su disco e recuperarli quando necessario; vedremo anche come mandare in stampa il risultato di un vostro programma.

CREAZIONE E USO DI FILE SEQUENZIALI

Un file è semplicemente una raccolta di dati conservata su disco. Un tipo di file che potete creare dall'interno di un programma scritto in QBasic è un *file sequenziale.* I contenuti di un file sequenziale devono essere usati dall'inizio alla fine in un ordine specifico (diversamente dai *file ad accesso casuale*, che non analizzeremo in questo libro, i cui contenuti possono essere usati in qualsiasi ordine). Per creare e lavorare con file sequenziali si usano gli enunciati e le funzioni seguenti:

Enunciato/Funzione	Descrizione
OPEN	Apre un file
CLOSE	Chiude un file
PRINT#	Stampa in un file dati non formattati
PRINT# USING	Stampa in un file dati formattati
WRITE#	Stampa in un file dati organizzati in campi
INPUT#	Recupera dati da un file
EOF	Controlla la fine di un file
LINE INPUT#	Recupera un'intera riga di dati da un file

In questo capitolo analizzeremo questi enunciati e queste funzioni in dettaglio per imparare a usare file sequenziali per registrare diversi tipi di informazioni (da un semplice diario ai contenuti di un complesso database).

NOTA: Per semplificare il processo di apprendimento, i file di questo capitolo verranno creati nell'unità e nella directory corrente (senza cioè l'uso di percorsi per file e directory). Naturalmente QBasic è pienamente compatibile col sistema di gestione di file e directory del DOS.

Creazione e apertura di un file

L'enunciato OPEN svolge una doppia funzione: viene usato sia per creare nuovi file che per aprire file esistenti. La sintassi dell'enunciato OPEN si presenta in questo modo:

```
OPEN nomefile FOR modalità AS #numerofile
```

nomefile è un qualsiasi nome di file valido per il DOS, *modalità* è una parola che indica come il file dev'essere usato e *numerofile* è un intero da 1 a 255 associato al file che viene aperto.

NOTA: Siccome a ciascun file viene assegnato un numero questo può essere usato per chiamare il file all'interno del programma. È consigliabile numerare i file all'interno del programma partendo da 1. Notate infine che l'assegnazione dura fino alla chiusura del file.

Specificare una modalità

Le modalità in cui un file può essere aperto sono tre, ma se ne può impiegare una sola alla volta; l'argomento *modalità* serve a specificare come si intende usare il file sequenziale. Gli argomenti utilizzabili per *modalità* in un enunciato OPEN sono i seguenti:

Modalità	Descrizione
OUTPUT	Crea e apre un file che riceverà un output dal programma. Se il file esiste già, i contenuti precedenti vengono cancellati.
APPEND	Apre un file esistente a cui viene aggiunto un output del programma; l'output, cioè, viene aggiunto alla fine del file. I contenuti originali del file vengono conservati.
INPUT	Apre un file che il programma può solo leggere ma non modificare. Il programma non può cambiare il file ma può usarne il contenuto come input.

Vedremo ora alcuni esempi di enunciati OPEN che mostrano l'uso di ciascuna modalità.

L'enunciato che segue apre il file sequenziale NOMI.TXT, che memorizza un output proveniente dal programma. Se questo file non esiste QBasic lo crea; se esiste già QBasic ne cancella i contenuti. NOMI.TXT è associato al numero 1.

```
OPEN "NOMI.TXT" FOR OUTPUT AS #1
```

L'enunciato che segue apre il file esistente NOMI.TXT e lo associa al numero 1. Il programma colloca il suo output alla fine dei contenuti esistenti di NOMI.TXT.

```
OPEN "NOMI.TXT" FOR APPEND AS #1
```

L'ultimo enunciato apre il file esistente NOMI.TXT e lo associa col numero 1. Il programma può usare i contenuti di NOMI.TXT come input.

```
OPEN "NOMI.TXT" FOR INPUT AS #1
```

Uso di una variabile stringa con OPEN

L'enunciato OPEN permette di usare anche una variabile stringa al posto di *nomefile*. Per esempio è possibile chiedere all'utente di fornire il nome di un file che poi viene assegnato a una variabile e usare quest'ultima all'interno dell'enunciato OPEN, come mostra l'esempio seguente:

```
INPUT "Che file desiderate aprire?  ", nomefile$
OPEN nomefile$ FOR OUTPUT AS #1
```

Può essere molto utile usare ciò che avete appreso nel capitolo 9 per far in modo che il programma verifichi che la risposta dell'utente (*nomefile$*) sia valida per il DOS prima di provare ad aprire il file.

Chiusura di un file

Quando si è terminato di usare un file bisogna assicurarsi che questo venga chiuso con l'enunciato CLOSE. In questo modo tutte le informazioni raccolte nel file vengono effettivamente salvate su disco. Dopo la chiusura, il numero associato al file può essere assegnato a un altro file.

La sintassi dell'enunciato CLOSE è la seguente:

CLOSE [*#numerofile*]

numerofile è il numero associato al file da chiudere; ricordatevi che se questo argomento non viene specificato tutti i file vengono chiusi. Per usare nuovamente un file dopo la chiusura lo si deve riaprire nei modi specificati in precedenza.

Ecco altri esempi relativi all'enunciato CLOSE. L'enunciato che segue chiude il file 1:


```
CLOSE #1
```

Quest'altro enunciato chiude tutti i file:

```
CLOSE
```

NOTA: Anche se QBasic al momento dell'uscita, chiude automaticamente tutti i file aperti, è consigliabile abituarsi a chiudere tutti i file non appena si è terminato di utilizzarli altrimenti si corre il rischio che il programma perda dei dati in caso di eventi imprevisti come la mancanza di corrente.

Archiviare informazioni in un file

Quando un file è stato aperto in modalità OUTPUT o APPEND è in grado di ricevere informazioni dal programma. Vi sono tre enunciati di QBasic che possono inviare informazioni a un file sequenziale aperto:

- L'enunciato PRINT# invia a un file dati *non formattati*.
- L'enunciato PRINT# USING invia a un file dati *formattati*.
- L'enunciato WRITE# invia a un file dati organizzati in *campi*.

I paragrafi che seguono descrivono questi tre enunciati in dettaglio.

L'enunciato PRINT#

L'enunciato PRINT# è funzionalmente simile a PRINT ma invece di inviare dati allo schermo li invia a un file. La sintassi dell'enunciato PRINT# è la seguente:

PRINT *#numerofile*, [*espressioneLista*] [, | ;]

numerofile è il numero del file aperto; *espressioneLista* è l'informazione da inviare al file. Se si omette quest'ultimo argomento QBasic invia al file una riga vuota.

L'argomento *espressioneLista* può essere composto da più elementi, purché siano separati da virgole o punti e virgola (questi svolgono esattamente la stessa funzione che svolgono nel consueto enunciato PRINT. Se un punto e virgola è l'ultimo carattere contenuto nell'enunciato PRINT#, il successivo elemento inviato al file appare alla fine della riga che avete appena inviato.

Esercizio: Archiviazione di dati non formattati in un file

Il programma AUTODATI.BAS (figura 10-1) mostra come usare l'enunciato PRINT# per inviare tre righe di dati non formattati a un file dal nome AUTODATI.TXT. I valori inviati a un file possono essere valori letterali, variabili semplici o il risultato di funzioni o espressioni.

Scrivete ed eseguite il programma AUTODATI.BAS.

Il risultato ottenuto dovrebbe essere simile a questo:

```
' AUTODATI.BAS
' Questo programma usa l'enunciato PRINT# per inviare tre righe
' di informazione sulle auto a un file sequenziale.

OPEN "AUTODATI.TXT" FOR OUTPUT AS #1 ' apre il file nell'unitã/directory
                                     ' corrente

CLS

' riceve informazioni sulle auto dall'utente e le scrive nel file aperto

INPUT "Inserire la marca di una macchina nella vostra raccolta: ",
marcaNome$
INPUT "Qual è il nome del modello? " modelloNome$
INPUT "Qual è l'anno di fabbricazione? " anno%

PRINT #1, marcaNome$, modelloNome$, anno%

' aggiunge al file dei valori letterali

PRINT #1, "Mercedes-Benz", "190 SL", 1959

' aggiunge una nuova macchina al file
' (la funzione RIGHT$ restituisce l'anno
' corrente dalla funzione DATE$).

PRINT #1, "Audi", "80 Quattro", " "; RIGHT$(DATE$, 4)

CLOSE #1     ' chiude il file

PRINT
PRINT "L'informazione è stata trasferita con successo al file
AUTODATI.TXT"
```

Figura 10-1. *Un programma che mostra l'uso dell'enunciato PRINT#.*


```
Inserire la marca di una macchina della vostra collezione: Alfa Romeo
Qual è il nome del modello? Giulietta
Qual è l'anno di fabbricazione? 1956

L'informazione è stata trasferita con successo al file AUTODATI.TXT
```

Un file dal nome AUTODATI.TXT viene creato nella directory e nell'unità corrente. Per visionare il file basta uscire da QBasic e scrivere il seguente comando DOS:

```
type autodati.txt
```

Il risultato sarà simile a questo:

```
Alfa Romeo          Giulietta           1956
Mercedes-Benz       190 SL              1959
Audi                80 Quattro          1991
```

L'enunciato PRINT# USING

L'enunciato PRINT# USING segue le stesse regole e usa gli stessi caratteri speciali di formattazione dell'enunciato PRINT USING ma invia dati formattati a un file invece che allo schermo. PRINT# USING è utile quando si vogliono inviare a un file grandi quantità di dati tabulari. La sintassi di questo enunciato è la seguente:

```
PRINT #numerofile, USING modello; [espressioneLista][,|;]
```

numerofile è il numero del file aperto, *modello* è una stringa usata per formattare i valori di *espressioneLista*, ed *espressioneLista* è il dato da formattare e da inviare al file (si possono usare virgole o punti e virgola per separare i valori in *espressioneLista*). I caratteri di formattazione nel *modello* devono essere in relazione uno a uno con i valori presenti in *espressioneLista.*

Esercizio: Conservazione di dati formattati in un file

Il programma FRUTTA.BAS (figura 10-2) mostra come usare l'enunciato PRINT# USING per inviare tre righe di dati formattati al file FRUTTA.TXT. Il modello di formattazione in questo programma è la variabile stringa *tmp$* che usa vari caratteri speciali di formattazione per allineare i dati verticalmente prima di inviarli al file.

Scrivete ed eseguite il programma FRUTTA.BAS.

```
' FRUTTA.BAS
' Questo programma usa l'enunciato PRINT# USING per inviare tre
' righe di informazioni formattate a un file sequenziale.

OPEN "FRUTTA.TXT" FOR OUTPUT AS #1' apre il file
nell'unità/directory corrente

CLS

' crea un modello di formattazione per l'enunciato PRINT# USING

tmp$ = "Frutta:  \     \  Casse: ###    Prezzo/Chilo:  L$##.###"

' riceve le informazioni dall'utente e trasferisce nel file aperto

INPUT "Scrivete il vostro frutto preferito: ", frutta$
INPUT "Quante casse volete acquistarne? ", casse%
INPUT "Quanto costa al chilo? L", costo!

PRINT #1, USING tmp$; frutta$; casse%; costo!

' aggiunge al file dei valori letterali

PRINT #1, USING tmp$; "Ciliege"; 2; 10.000

PRINT #1, USING tmp$; "Melone"; 14; 9.500

CLOSE #1    ' chiude il file

PRINT
PRINT "L'informazione è stata trasferita con successo al file
FRUTTA.TXT"
```

Figura 10-2. *Un programma che mostra l'uso di PRINT# USING.*

Il risultato che otterrete sarà simile a questo:

```
Scrivete il vostro frutto preferito: Pera
Quante casse volete acquistarne? 10
Quanto costa al chilo? L5.300

L'informazione è stata trasferita con successo al file FRUTTA.TXT
```


Un file dal nome FRUTTA.TXT viene creato nell'unità e nella directory corrente; per esaminarlo basta uscire da QBasic e digitare il seguente comando DOS:

```
type frutta.txt
```

L'output ottenuto dovrebbe essere simile a questo:

```
Frutta:  Pera       Casse:  10    Prezzo/Chilo:   L5.300
Frutta:  Ciliege    Casse:   2    Prezzo/Chilo:   L10.000
Frutta:  Melone     Casse:  14    Prezzo/Chilo:  L9.500
```

L'enunciato WRITE#

L'enunciato WRITE# non formatta il proprio output come fanno gli enunciati PRINT e PRINT# USING ma crea un file che potrà essere letto da altri programmi (questo enunciato verrà usato più avanti in questo capitolo per la creazione di file di database). L'informazione che l'enunciato WRITE# invia a un file sequenziale è separata da *virgole* ed è organizzata in gruppi, detti *campi*.

La sintassi di WRITE# si presenta in questo modo:

```
WRITE #numerofile[, espressioneLista]
```

numerofile è il numero del file aperto; *espressioneLista* rappresenta le informazioni da inviare al file (gli elementi in essa contenuti vengono separati da virgole). Ricordate che se non si inserisce l'argomento *espressioneLista*, il file riceverà una semplice riga vuota.

La figura 10-3 mostra un esempio di enunciato WRITE# che invia a un file cinque tipi di valori. Notate come le virgole tra i valori separino l'output nei campi; notate anche che il valore stringa si trova tra doppi apici. Infatti, se la stringa contenesse spazi o virgole, un programma non in QBasic li identificherebbe come elementi della stringa). Ogni riga del file creato con WRITE# viene detta *registro* (record).

Esercizio: Archiviazione di dati in un file con i campi

Il programma MONETE.BAS (figura 10-4) mostra come usare l'enunciato WRITE# per archiviare informazioni su una collezione di monete in un file detto MONETE.TXT. MONETE.BAS è creato in modo da consentire all'utente l'immissione di informazioni su un numero desiderato di monete. Per

Un esempio di programma:

```
OPEN "TESTO.TXT" FOR OUTPUT AS #1        ← Enunciato OPEN

a$ = "Giuseppe"
b% = 25
c& = 100000                              ← Variabili di esempio
d! = 115.5
e# = .0123456789#

WRITE #1, a$, b%, c&, d!, e#             ← Enunciato WRITE#
CLOSE #1                                 ← Enunciato CLOSE
```

Output inviato al file:

```
"Giuseppe", 25, 100000, 115.5, .0123456789#
```

Campo 1 Campo 2 Campo 3 Campo 4 Campo 5

Figura 10-3. *L'enunciato WRITE# archivia i dati in un file in campi e registri.*

interrompere l'inserimento di dati basterà scrivere FINE alla richiesta di informazioni sulla nazionalità della moneta.

Scrivete ed eseguite il programma MONETE.BAS.

Il risultato dovrebbe essere simile a questo:

```
Questo programma archivia le informazioni sulla collezione di monete
in un file dal nome MONETE.TXT. Inserire i dati; premere FINE per
uscire.

Da quale paese proviene la moneta? Stati Uniti
Qual è il valore della moneta? 10 centesimi
Qual è il nome della moneta? Dime
In quale anno è stata emessa? 1980

Da quale paese proviene la moneta? Canada
Qual è il valore della moneta? 25 centesimi
Qual è il nome della moneta? Quarter
In quale anno è stata emessa? 1960

Da quale paese proviene la moneta? Ungheria
Qual è il valore della moneta? 2 forints
```



```
' MONETE.BAS
' Questo programma usa l'enunciato WRITE# per inviare a un file
' sequenziale informazioni su una collezione di monete,
' suddivise in campi.

OPEN "MONETE.TXT" FOR OUTPUT AS #1     ' apre il file nell'unità/
                                       ' directory corrente

CLS

PRINT "Questo programma archivia le informazioni sulla collezione di
monete"
PRINT "in un file dal nome MONETE.TXT. Inserire i dati; digitare FINE
per uscire."
PRINT
DO WHILE (nazione$ ,. "FINE")   ' Esegue finché l'utente digita FINE...

' riceve le informazioni dall'utente e le trasferisce nel file aperto

      INPUT "Da quale nazione proviene la moneta? ", nazione$
      IF (nazione$ <> "FINE") THEN    ' se nazione$ è FINE non
                                      ' trasferisce dati nel file
            INPUT "Qual è il valore della moneta? ", valore$
            INPUT "Qual è il nome della moneta? ", nomeMoneta$
            INPUT "In quale anno è stata emessa? ", anno%

      WRITE #1, nazione$, valore$, nomeMoneta$, anno%    ' Invia i campi
      END IF
      PRINT ' stampa una riga bianca tra le monete

LOOP

CLOSE #1     ' chiude il file

PRINT "L'informazione è stata trasferita con successo al file MONETE.TXT"
```

Figura 10-4. *MONETE.BAS. Un programma che dimostra l'enunciato WRITE#.*

```
Qual è il nome della moneta?
In quale anno è stata emessa? 1955

Da quale paese proviene la moneta? Gran Bretagna
Qual è il valore della moneta? 1 sterlina
Qual è il nome della moneta? Pound
```

```
In quale anno è stata emessa? 1981

Da quale paese proviene la moneta? FINE

L'informazione è stata trasferita con successo al file MONETE.TXT
```

Un file dal nome MONETE.TXT viene creato nell'unità e nella directory corrente; per verificarlo basta uscire da QBasic e digitare il seguente comando DOS:

```
type monete.txt
```

L'output che otterrete sarà simile a questo:

```
"Stati Uniti","10 centesimi","Dime",1980
"Canada","25 centesimi","Quarter",1960
"Ungheria","2 forints","",1955
"Gran Bretagna","1 sterlina","Pound",1981
```

Il file MONETE.TXT contiene quattro registri composti di quattro campi ciascuno, come mostrato nella figura 10-5.

Impiego di un file come input di dati

Dopo aver visto come creare tre diversi tipi di file e come esaminarli attraverso il comando TYPE del DOS, cercheremo ora di spiegare come esaminare e usare questi file di dati all'interno di un programma in QBasic. In particolare prenderemo in esame due enunciati e una funzione che servono per lavorare con file aperti per avere informazioni:

- L'enunciato INPUT# riceve da un file *uno o più campi* di dati.
- La funzione EOF determina se è stata raggiunta la fine del file.
- L'enunciato LINE INPUT# riceve da un file un'intera riga di dati.

L'enunciato INPUT#

L'enunciato INPUT# è il principale strumento per ottenere dati da un file sequenziale di QBasic. Questo enunciato riceve l'input da un file sequenziale nello stesso modo in cui l'enunciato INPUT lo riceve dalla tastiera. Entrambi questi enunciati assegnano uno o più elementi di informazione a variabili dello stesso tipo. La sintassi di INPUT# è la seguente:

```
INPUT #numerofile, variabileLista
```


Figura 10-5. *Ogni riga contenuta nel file MONETE.TXT è un registro composto di quattro campi.*

numerofile è il numero del file aperto; *variabileLista* indica le variabili cui vengono assegnati i dati ricevuti dal file. Gli elementi contenuti nell'argomento *variabileLista* devono essere dello stesso tipo di quelli che si trovano nel file sequenziale; questi possono essere campi di dati creati con l'enunciato WRITE# o output dell'enunciato PRINT#, PRINT# USING o di qualsiasi altro programma in grado di creare file di dati.

Esercizio: Lettura di elementi di informazione da un file sequenziale

Il programma AMICI.BAS (figura 10-6) mostra come usare l'enunciato INPUT# per ottenere dati da un file sequenziale. Il programma che segue apre un file sequenziale nella directory corrente in modalità output e vi immette quattro stringhe, usando WRITE#; poi chiude il file, lo riapre in modalità input e riporta le quattro stringhe nel programma.

Scrivete ed eseguite il programma AMICI.BAS.

Il risultato ottenuto sarà simile a questo:

```
Scrivete il nome di quattro vostri amici

Nome: Andrea
Nome: Sandro
Nome: Fabio
Nome: Davide

Avete inserito i nomi seguenti:

Andrea
Sandro
Fabio
Davide
```

Il file AMICI.TXT creato sul disco conterrà i seguenti registri di dati:

```
' AMICI.BAS
' Questo programma mostra l'uso dell'enunciato INPUT#

CLS

OPEN "AMICI.TXT" FOR OUTPUT AS #1       ' apre il file come output

PRINT "Scrivete il nome di quattro vostri amici"
PRINT

FOR i% = 1 TO 4     ' entra in un loop per quattro volte
      INPUT "Nome: ", amico$    ' ogni volta riceve un nome
                                ' dall'utente...
      WRITE #1, amico$   ' e lo trasferisce sul disco
NEXT i%

CLOSE #1     ' chiude il file

OPEN "AMICI.TXT" FOR INPUT AS #1       ' riapre il file come input

PRINT
PRINT "Avete inserito i nomi seguenti:"
PRINT

FOR i% = 1 TO 4              ' entra in un loop per quattro volte
      INPUT #1, amico$       ' ogni volta legge un nome dal file...
      PRINT amico$           ' e lo mostra sullo schermo
NEXT i%

CLOSE #1     Chiude il file
```

Figura 10-6. *Un programma che mostra l'uso dell'enunciato INPUT#.*

```
"Andrea"
"Sandro"
"Fabio"
"Davide"
```

La funzione EOF

Se il programma in QBasic non è in grado di determinare dove finisce il file sequenziale si può ricevere un messaggio di errore *end-of-file* (fine del file); questo errore si verifica quando QBasic cerca di leggere oltre la fine del file.

Per evitare questo errore basta indicare a QBasic di terminare la lettura dei valori quando incontra la fine del file; a questo scopo si usa la funzione EOF, che presenta questa sintassi:

`EOF(`*`numerofile`*`)`

numerofile è il numero del file aperto che si vuole controllare.

EOF restituisce il valore logico *vero* se il successivo carattere da leggere si trova oltre la fine del file e *falso* nel caso opposto.

È possibile usare la funzione EOF in un qualsiasi punto del programma per controllare un file aperto, ma la sua collocazione ottimale è in un loop DO, come evidenziato dall'esercizio che segue.

Esercizio: Individuazione della fine di un file

Il programma MONETE2.BAS (figura 10-7) è un miglioramento del programma MONETE.BAS visto in precedenza. Il programma usa l'enunciato INPUT# per visualizzare i dati del file e la funzione EOF per controllare la fine del file. La nuova versione del programma apre il file che contiene i dati sulla collezione di monete in modalità APPEND in modo da evitare che le informazioni precedentemente raccolte vengano sovrascritte da quelle nuove; quest'ultime vengono collocate alla fine del file.

Scrivete ed eseguite il programma MONETE2.BAS

Al momento dell'esecuzione il programma richiederà le informazioni nel modo seguente:

```
Questo programma raccoglie le informazioni su una collezione di
monete su disco in un file dal nome MONETE.TXT. Inserire i dati;
digitare FINE per uscire.

Da quale paese proviene la moneta? Olanda
Qual è il valore della moneta? 2.5 guilders
Qual è il nome della moneta? Rijksdaalder
In quale anno è stata emessa? 1982

Da quale paese proviene la moneta? FINE
```

```
' MONETE2.BAS
' Questo programma usa l'enunciato WRITE# per inviare informazioni sulla
' collezione di monete a un file sequenziale e INPUT# per visualizzarle.

' apre il file in modalità APPEND per non sovrascrivere dati precedenti

OPEN "MONETE.TXT" FOR APPEND AS #1      ' apre il file nell'unità/directory
                                        'corrente

CLS

PRINT "Questo programma raccoglie le informazioni su una collezione di
monete su
PRINT "disco in un file dal nome MONETE.TXT. Inserire i dati; digitare
FINE per uscire."
PRINT

DO WHILE (nazione$ <> "FINE")   ' fino a che l'utente digita FINE...

' raccoglie le informazioni dall'utente e le trasferisce nel file aperto

        INPUT "Da quale paese proviene la moneta? ", nazione$
        IF (nazione$ <> "FINE") THEN     ' Se nazione$ è FINE
                                         ' non trasferisce dati nel file
                INPUT "Qual è il valore della moneta? ", valore$
                INPUT "Qual è il nome della moneta? ", nomeMoneta$
                INPUT "In quale anno è stata emessa? ", anno%

        WRITE #1, paese$, valore$, nomeMoneta$, anno%       ' invia i campi
        END IF

        PRINT ' Stampa una riga bianca tra le monete

LOOP

CLOSE #1      ' chiude il file

' aspetta che l'utente prema Invio per continuare
```

Figura 10-7. *Un programma che mostra l'uso della funzione EOF e dell'enunciato INPUT# (continua).*


```
INPUT "Premere Invio per vedere il contenuto della collezione", istruz$
CLS    ' inizia con uno schermo vuoto

' apre il file in modalità INPUT cosi' i contenuti possono venire
' letti dal programma

OPEN "MONETE.TXT" FOR INPUT AS #1       ' il file è nell'unità/
                                        ' directory corrente

' visualizza l'intestazione per la formattazione tabulare
' delle informazioni

PRINT"Origine        ValoreNome   Anno"
PRINT"-------------------------------------------------------------"
PRINT

' inizializza il modello di formattazione da usare con PRINT USING

tmp$ = "\     \  \   \  \   \  ####"

' se la fine del file non è stata raggiunta, il programma
' assegna gli elementi del file a variabili e li visualizza.

DO WHILE (NOT EOF(1))
        INPUT #1, nazione$, valore$, nomeMoneta$, anno%
        PRINT USING tmp$; nazione$; valore$; nomeMoneta$; anno%
LOOP

CLOSE #1      ' chiude il file
```

Figura 10-7. *Un programma che mostra l'uso della funzione EOF e dell'enunciato INPUT#.*

Dopo aver inserito le nuove informazioni queste vengono visualizzate sullo schermo insieme a quelle già conservate nel file:

```
Origine          Valore          Nome            Anno
-----------------------------------------------------------
Stati Uniti      10 centesimi    Dime            1980
Canada           25 centesimi    Quarter         1960
Ungheria          2 forints                      1955
Gran Bretagna     1 sterlina     Pound           1981
Olanda            2.5 guilders   Rijksdaalder    1982
```

L'enunciato LINE INPUT

L'enunciato INPUT# è un modo efficace per leggere elementi singoli dai file; se però si vogliono trasferire lunghe stringhe di informazioni testuali, QBasic mette a disposizione l'enunciato LINE INPUT#. L'enunciato è simile a LINE INPUT con la differenza che riceve il proprio input da un file invece che dalla tastiera. La sintassi dell'enunciato LINE INPUT# è la seguente:

```
LINE INPUT #numerofile, nomevariabile$
```

numerofile è il numero del file aperto da cui trasferire le informazioni; *nomevariabile$* è il nome della variabile stringa che deve ricevere l'input.

L'enunciato LPRINT

Per inviare informazioni a una stampante collegata al computer si usa l'enunciato LPRINT, la cui sintassi è simile a quella dell'enunciato PRINT:

```
LPRINT [espressioneLista] [,|;]
```

espressioneLista è una lista di espressioni stringa o numeriche separate da *virgole* o *punti e virgola*. Per esempio, per inviare alla stampante la stringa *Buon Natale!* basta usare questo enunciato LPRINT:

```
LPRINT "Buon Natale!"
```

Per inviare alla stampante un comando di formattazione (per esempio il comando di avanzamento della carta) una volta terminata la stampa servitevi dell'enunciato seguente:

```
LPRINT CHR$(12)
```

Questo enunciato invia alla stampante il carattere ASCII 12 per far avanzare la carta.

Esercizio: Lettura di un diario con LINE INPUT#

Il programma DIARIO.BAS (figura 10-8) mostra come usare gli enunciati LINE INPUT e LINE INPUT# per scrivere ampi brani di testo in un file sequenziale. DIARIO.BAS viene prima aperto in modalità APPEND e viene poi marcato con l'ora e la data corrente dall'orologio di sistema. L'utente può aggiungere una o più righe al diario che viene poi conservato nel file DIARIO.TXT nella directory corrente. Poiché il file del diario viene aperto in modalità APPEND ogni informazione viene aggiunta alla fine del file, senza che il contenuto precedente vada perso. Se l'utente richiede una stampa, tutte le informazioni (inclusa la data e l'ora) vengono inviate alla stampante dagli enunciati LPRINT.

Scrivete ed eseguite il programma DIARIO.BAS.

Al momento dell'esecuzione il programma richiede di completare il diario nel modo seguente:

```
Scrivete i vostri pensieri giornalieri; digitate FINE per uscire.
```

Dopo aver digitato le informazioni e la parola *FINE*, il programma effettua la seguente richiesta:

```
Desiderate stampare il diario su carta (S,N)?
```

Se la risposta è Sì, il programma invia alla stampante una copia del diario.

```
' DIARIO.BAS
' Questo programma mantiene un diario computerizzato
' in un file sequenziale detto DIARIO.TXT

CLS

OPEN "DIARIO.TXT" FOR APPEND AS #1      ' apre il file in modalità append

PRINT "Scrivete i vostri pensieri giornalieri; digitate FINE per uscire."
PRINT

PRINT #1, TIME$; " "; DATE$     ' scrive nel file l'ora e la data
PRINT #1,    ' inserisce nel file una riga vuota

DO WHILE (UCASE$(riga$) <> "FINE")     ' fino a che l'utente digita "FINE"
     LINE INPUT riga$    ' prende le righe di testo...
     IF (riga$ <> "FINE") THEN PRINT #1, riga$
LOOP  ' e le trasferisce nel file

PRINT #1,    ' inserisce nel file una riga vuota
CLOSE #1     ' chiude il file

PRINT  ' chiede all'utente se desidera una stampa
INPUT "Desiderate stampare il diario su carta (S,N)? ", risposta$
```

Figura 10-8. *Un programma per un semplice diario che mostra l'uso di LINE INPUT# (continua).*

```
IF (UCASE$(risposta$) = "S") THEN      ' se la risposta è Si...
     OPEN "DIARIO.TXT" FOR INPUT AS #1      ' apre il file di input

     LPRINT STRING$(33, "-"); ' stampa un'intestazione in cima
                              ' alla pagina...
     LPRINT " Diario Personale ";
     LPRINT STRING$(##, "-")
     LPRINT ' e una riga vuota

     DO WHILE (NOT EOF(1))      ' fino a quando la fine del file
                                ' non viene raggiunta..
          LINE INPUT #1, riga$      ' legge le righe dal file
          LPRINT riga$              ' e le invia alla stampante
     LOOP

     CLOSE #1      ' chiude il file

     LPRINT CHR$(12)      ' invia il carattere di formattazione

     PRINT ' mostra il messaggio che indica che...
     PRINT "Diario in fase di stampa"      ' i contenuti del diario
                                           , sono in fase di stampa
END IF
```

Figura 10-8. *Un programma per un semplice diario che mostra l'uso di LINE INPUT#.*

L'enunciato VIEW PRINT

L'ultimo enunciato che vedremo in questo paragrafo è VIEW PRINT; l'enunciato non viene usato direttamente coi file ma spesso viene impiegato in combinazione con INPUT#, LOCATE e PRINT per visualizzare il contenuto di un file in posti diversi dello schermo. Fino ad ora abbiamo usato per l'output di enunciati come PRINT l'intero schermo; quando questo è completo altri enunciati PRINT lo fanno scorrere. Con l'enunciato VIEW PRINT è possibile restringere l'output solo a una parte dello schermo (detta finestra di testo) mantenendo inalterato il resto. La sintassi di VIEWPRINT è la seguente:

```
VIEW PRINT [lineaSup TO lineaInf]
```

lineaSup e *lineaInf* rappresentano la linea superiore e quella inferiore della finestra di testo; se queste due informazioni non vengono specificate la finestra di testo corrisponde all'intero schermo.

Una volta definita la finestra di testo, lo scorrimento avviene solamente tra la linea superiore e inferiore della finestra. Ciò può essere molto utile quando si vuole visualizzare un elenco di istruzioni o le intestazioni di una tabella evitando che queste escano dallo schermo. Notate che quando una finestra di testo è attiva si può usare l'enunciato

```
CLS 2
```

per ripulire il contenuto della finestra stessa, lasciando intatto il resto dello schermo.

Esercizio: Creazione di una finestra di testo

Il programma VIEW.BAS (figura 10-9) mostra l'uso degli enunciati VIEW PRINT, LOCATE, PRINT e CLS. Maggiori informazioni su questi enunciati verranno fornite più oltre quando vedremo come creare un programma per un database.

Se eseguite questo programma potrete vedere lo schermo con una finestra di testo divisa, come si vede nella figura seguente.

```
Benvanuti nel programma VIEW!

Osserviamo lo scorrere di alcune informazioni...
--------------------------------------------------------------------------------
Questa è la riga 183
Questa è la riga 184
Questa è la riga 185
Questa è la riga 186
Questa è la riga 187
Questa è la riga 188
Questa è la riga 189
Questa è la riga 190
Questa è la riga 191
Questa è la riga 192
Questa è la riga 193
Questa è la riga 194
Questa è la riga 195
Questa è la riga 196
Questa è la riga 197
Questa è la riga 198
Questa è la riga 199
Questa è la riga 200

--------------------------------------------------------------------------------
Premere Invio per pulire la finestra di testo...
```

```
' VIEW.BAS
' Questo programma mostra l'uso dell'enunciato VIEW PRINT

CLS

PRINT "Benvenuti nel programma VIEW!!"        ' stampa il messaggio di
                                              ' introduzione
PRINT
PRINT "Osserviamo lo scorrere di alcune informazioni..."
PRINT STRING$(80, "-");  ' stampa la linea superiore

LOCATE 24, 1: PRINT STRING$980, "-"); ' stampa la linea inferiore
LOCATE 25, 1: INPUT ; "Premere Invio per avviare il programma",
richiesta$

VIEW PRINT 5 TO 23 ' attiva la finestra di testo (righe 5-23)

COLOR 2       ' imposta il colore verde
FOR i% = 1 TO 200
     PRINT "Questa è la riga"; i%    ' stampa il messaggio 200 volte
NEXT i%
COLOR 7       ' imposta il colore bianco predefinito

VIEW PRINT  ' disattiva la finestra di testo
LOCATE 25, 1: INPUT ; "Premere Invio per pulire la finestra di
testo...", richiesta$

VIEW PRINT 5 TO 23 ' Attiva la finestra di testo
CLS 2 ' pulisce solo la finestra di testo
```

Figura 10-9. *Un programma che mostra l'uso dell'enunciato VIEW PRINT.*

Enunciati di QBasic equivalenti a comandi di DOS

Molti enunciati di QBasic hanno delle controparti nel mondo DOS. Se avete già una certa conoscenza dei comandi DOS per la gestione dei file troverete sicuramente utile la tabella 10-1; essa elenca gli enunciati di QBasic con i correlati comandi di DOS e con una descrizione della loro funzione.

Tabella 10-1. *Enunciati di QBasic e comandi DOS equivalenti*

Enunciato di QBasic	Equivalente DOS	Descrizione
FILES	DIR/W	Mostra il contenuto di una directory
SHELL	COMMAND	Esegue un comando di DOS
NAME	RENAME	Cambia il nome di un file
KILL	DEL o ERASE	Cancella un file
MKDIR	MKDIR	Crea una directory
CHDIR	CHDIR	Cambia la directory corrente
RMDIR	RMDIR	Rimuove una directory vuota

L'enunciato FILES

Per far in modo che il programma mostri un elenco di file tra cui l'utente può scegliere si usa l'enunciato FILES, che presenta la seguente sintassi:

`FILES [nomefile]`

nomefile è il nome facoltativo del file che si desidera vedere nella directory che lo contiene; se non viene specificato, QBasic mostra un elenco di tutti i file presenti nella directory corrente. I caratteri jolly * e ? possono essere usati all'interno di *nomefile* per visualizzare gruppi di file più ampi. Per vedere file contenuti in un'unità disco o directory diversa da quella corrente bisogna includere in *nomefile* la lettera che indica l'unità disco o il percorso (path) della directory desiderata.

Esercizio: Visualizzazione della directory corrente

Il programma MOSTRA.BAS (figura 10-10) mostra come usare il comando FILES per visualizzare tutti i file con estensione .TXT contenuti nella directory corrente. Dopo aver visualizzato l'elenco dei file MOSTRA.BAS, chiede all'utente di specificare il nome di un file e lo stampa.

Scrivete ed eseguite il programma MOSTRA.BAS.

```
' MOSTRA.BAS
' Questo programma permette di vedere un file di dati
' della directory corrente.

CLS

PRINT "La directory corrente contiene i seguenti file";
PRINT "con estensione .TXT:"
PRINT

FILES "*.TXT"

PRINT
INPUT "Quale file desiderate aprire? ", nomefile$

OPEN nomefile$ FOR INPUT AS #1

PRINT
PRINT"------------ "; UCASE$(nomefile$); " ------------"
PRINT

DO WHILE (NOT EOF(1))
     LINE INPUT #1, linea$
     PRINT linea$
LOOP
```

Figura 10-10. *Un programma che mostra l'uso dell'enunciato FILES.*

Al momento dell'esecuzione otterrete un risultato simile a questo:

```
La directory corrente contiene i seguenti file con estensione .TXT:

C:\DOS
AUTODATI.TXT FRUTTA.TXT   MONETE.TXT   AMICI.TXT    DIARIO.TXT
  2080768 Bytes liberi

Quale file desiderate aprire? frutta.txt

----------FRUTTA.TXT----------

Frutta:  Pera      Casse:  10   Prezzo/Chilo:   L5.300
Frutta:  Ciliege   Casse:  2    Prezzo/Chilo:   L10.000
Frutta:  Melone    Casse:  14   Prezzo/Chilo:   L9.500
```


L'enunciato SHELL

L'enunciato SHELL permette di uscire temporaneamente da QBasic per eseguire dei comandi DOS; è utile quando si vuole eseguire un comando non disponibile in QBasic o quando si vuole dare all'utente accesso al DOS in fase di esecuzione di un programma (questa funzionalità si trova in molti programmi presenti oggi sul mercato).

In Microsoft Windows, per esempio, basta selezionare l'icona del DOS per entrarvi temporaneamente.

L'enunciato SHELL presenta questa sintassi:

```
SHELL [comandoStringa]
```

comandoStringa è un valore o una variabile stringa facoltativa che contiene un comando DOS; se questo argomento viene incluso DOS esegue il comando e ritorna nel programma in QBasic; in caso contrario, si può dire che DOS esegua temporaneamente una copia di se stesso, copia che funziona esattamente come quella principale e in cui si può eseguire un qualsiasi numero di comandi. Tuttavia la copia è soltanto temporanea; basta infatti digitare *exit* e premere Invio per ritornare nel programma in QBasic.

Esercizio: Esecuzione di un comando di DOS

Il programma RUNDOS.BAS (figura 10-11) mostra come usare l'enunciato SHELL per eseguire dei comandi DOS dall'interno di un programma in QBasic. Per aiutarvi a distinguere ciò che il programma e il DOS visualizzano, il programma imposta un colore di fondo azzurro. In questo modo i risultati degli enunciati di QBasic appariranno in questo colore e i risultati del DOS appariranno nel colore bianco predefinito.

1. Scrivete ed eseguite il programma RUNDOS.BAS
   QBasic mostrerà la seguente richiesta:

   ```
   Inserite un comando DOS [premete Invio per eseguirlo]:
   ```

2. Per vedere come l'enunciato SHELL gestisce un comando DOS provate a digitare *chkdsk* e premete Invio. Dopo qualche momento il DOS mostra delle informazioni simili a queste:

   ```
   Volume LORENZO      creato 05-02-1990 4:36p
   Il numero di serie del volume è 1810-1496
   ```

```
' RUNDOS.BAS
' Questo programma esegue un comando DOS

CLS

COLOR 3         ' imposta il colore azzurro
                ' riceve un comando DOS dall'utente
INPUT "Inserite un comando DOS [premete Invio per eseguirlo]: ", dosCom$
PRINT

SHELL dosCom$       ' esce in DOS ed esegue il comando

PRINT
PRINT "Comando DOS completato"

COLOR &         ' Imposta il colore bianco predefinito
```

Figura 10-11. *Un programma che mostra l'uso dell'enunciato SHELL.*

```
21690368 byte totali su disco
77824 byte in 6 file nascosti
120832 byte in 54 directory
19410944 byte in 1003 file utente
2080768 byte disponibili su disco

2048 byte in ogni unità di allocazione
10591 unità di allocazione totali su disco
1016 unità di allocazione disponibili su disco

655360 byte di memoria totale
361888 byte liberi

Comando DOS completato
```

L'enunciato SHELL elabora il comando CHKDSK del DOS e poi torna nel programma.

3. Eseguite il programma di nuovo premendo Invio senza specificare alcun comando (eseguendo perciò una versione temporanea di DOS). Avrete un risultato simile a questo (notate il modo usato per copiare un file in DOS):


```
Inserite un comando DOS [premete Invio per eseguirlo]:

Microsoft(R) MS-DOS(R) Versione 5.00
  (C)Copyright Microsoft Corp 1981-1991.

C:\DOS>dir *.txt

Il volume nell'unità C è LORENZO
Il numero di serie del volume è 1810-1496
Directory di C:\DOS
AUTODATI.TXT       107  06-06-91  12:40p
FRUTTA.TXT         177  06-06-91  13:56p
MONETE.TXT         197  06-06-91  15:34p
AMICI.TXT           37  06-07-91  10:19a
DIARIO.TXT         713  06-07-91  13:12p
        5 file        1231 byte
                      2080768 byte liberi

C:\DOS>copy diario.txt a:
        1 File copiato

C:\DOS>exit

Comando DOS completato
```

L'enunciato NAME

L'enunciato NAME è utile quando si vuole rinominare un file o spostarlo da una directory a un'altra. La sintassi dell'enunciato NAME è la seguente:

```
NAME vecchioNomeFile AS nuovoNomeFile
```

vecchioNomeFile è il nome attuale del file e *nuovoNomeFile* è il nome che gli si vuole attribuire; in entrambi questi argomenti si può definire un percorso di directory; se i due percorsi sono diversi, QBasic sposta il file nella directory specificata da nuovoNomeFile. Questo enunciato non consente di spostare file tra unità disco diverse. Se, per qualche ragione, NAME non riesce a rinominare i file, il programma genera un messaggio di errore.

Esercizio: Rinominare un file

Il programma RINOMINA.BAS (figura 10-12) mostra l'uso dell'enunciato NAME per cambiare il nome di un file nell'unità corrente. Notate l'uso dell'enunciato FILES che mostra i file contenuti della directory corrente.

Scrivete ed eseguite il programma RINOMINA.BAS.

```
' RINOMINA.BAS
' Questo programma permette di rinominare un file
' nella directory corrente.

CLS

PRINT "La directory corrente contiene i seguenti file:"
PRINT

FILES "*.*" ' mostra i file nella directory corrente

PRINT ' riceve il nome attuale e quello nuovo
INPUT "Quale file vuoi rinominare? ", vecchioNome$
INPUT "Qual è il nuovo nome del file? ", nuovoNome$

NAME vecchioNome$ AS nuovoNome$    ' prova a rinominare il file

' se vecchioNome$ non esiste o nuovoNome$ non è valido
' l'enunciato NAME genera un messaggio di errore; altrimenti
' vengono eseguite le seguenti righe del programma:

PRINT       ' stampa il messaggio successivo
PRINT UCASE$(vecchioNome$); "rinominato con successo"
```

Figura 10-13. *Un programma che mostra l'uso dell'enunciato NAME.*

Il risultato ottenuto dall'esecuzione del programma sarà simile a questo:

```
La directory corrente contiene i seguenti file:

C:\DOS
    ..  <DIR>   ..  <DIR>       FRUTTA.BAS      AUTODATI.BAS
AUTODATI.TXT    FRUTTA.TXT      MONETE.BAS      MONETE.TXT
MONETE2.BAS     AMICI.BAS       AMICI.TXT       DIARIO.BAS
MOSTRA.BAS      RINOMINA.BAS    DIARIO.BAS      RUNDOS.BAS

2080013 Byte liberi

Quale file vuoi rinominare? diario.txt
Qual è il nuovo nome del file? libro.txt

DIARIO.TXT rinominato con successo
```

L'enunciato KILL

L'enunciato KILL ha un significato diretto e definitivo: cancella un file dal disco. Quando un file è stato cancellato con l'enunciato KILL non può più essere recuperato. I programmatori usano quest'enunciato generalmente per cancellare file temporanei creati durante l'esecuzione del programma, ma può essere usato anche per funzioni di pulizia del disco. La sintassi dell'enunciato KILL è la seguente:

`KILL nomefile`

nomefile è il nome del file che dev'essere cancellato. Con questo enunciato si possono dare indicazioni sull'unità, il percorso e si possono anche usare i caratteri jolly ? e *.

NOTA: L'enunciato KILL va usato con estrema cautela, specialmente se si impiegano dei caratteri jolly; si corre infatti il rischio di cancellare file che non si volevano cancellare. L'esercizio che segue mostra alcune misure di sicurezza che possono essere inserite in un programma che usa l'enunciato KILL.

Esercizio: Cancellazione di un file

Il programma KILL.BAS (figura 10-14) si serve dell'enunciato KILL per cancellare un file. Se l'utente include un carattere jolly nell'argomento *nomefile* il programma lo rileva per mezzo della funzione INSTR e visualizza un avvertimento in colore rosso che chiede all'utente di confermare l'azione che sta per essere eseguita. Se il file specificato non esiste, l'enunciato KILL genera un messaggio di errore.

Scrivete ed eseguite il programma KILL.BAS

```
' KILL.BAS
' Questo programma permette di cancellare un file nella directory
corrente
CLS
PRINT "La directory corrente contiene i seguenti file:"
PRINT
FILES "*.*" ' mostra i file nella directory corrente
PRINT ' riceve il file da cancellare
INPUT "Quale(i) file desideri cancellare? ", nomefile$

' controlla l'esistenza di caratteri jolly (? e *) in nomefile$
' se ve ne sono, mostra un messaggio di avvertimento
' se non ve ne sono, prosegue con la cancellazione
' se il file non esiste, mostra un messaggio di errore
IF (INSTR(nomefile$, "?")) OR (INSTR(nomefile$, "*")) THEN
        PRINT
        COLOR 4      ' imposta il colore rosso
        PRINT "Attenzione: i caratteri jolly possono far cancellare
        più file!"
        COLOR 7      ' imposta il colore bianco predefinito
        PRINT ' chiede conferma della cancellazione
        INPUT "Vuoi proseguire (S,N)? ", risposta$
        ' se la risposta è Si, i file vengono cancellati
        IF (UCASE$(risposta$) = "S") THEN
                KILL nomefile$
                PRINT
                PRINT UCASE$(nomefile$); " cancellato(i) dal sistema"
        END IF ' ...in caso contrario esce dal programma
ELSE
        KILL nomefile$  ' se non ci sono caratteri jolly, cancella i file
        PRINT
        PRINT UCASE$(nomefile$); " cancellato(i) dal sistema"
END IF
```

Figura 10-14. *Un programma che mostra l'uso dell'enunciato KILL.*

Il risultato ottenuto sarà simile a questo:

```
La directory corrente contiene i seguenti file:

C:\DOS
    ..  <DIR>    ..  <DIR>       FRUTTA.BAS     AUTODATI.BAS
AUTODATI.TXT     FRUTTA.TXT      MONETE.BAS     MONETE.TXT
MONETE2.BAS      AMICI.BAS       AMICI.TXT      DIARIO.BAS
MOSTRA.BAS       RINOMINA.BAS    DIARIO.BAS     RUNDOS.BAS

2080013 Byte liberi

Quale(i) file desideri cancellare? DIARIO.TXT

DIARIO.TXT cancellato(i) dal sistema
```

CREAZIONE DI UN DATABASE CON FILE SEQUENZIALI

Concludiamo questo capitolo con uno sguardo a un semplice programma di database, MUSICDB.BAS, che usa molte delle enunciati e delle funzioni viste per lavorare con informazioni conservate in un file sequenziale. MUSICDB.BAS registra le seguenti informazioni per ogni disco, cassetta e compact disc della vostra collezione:

- Titolo
- Artista
- Anno di pubblicazione sul mercato
- Stile (rock, blues, jazz, classica, etc.)
- Supporto (disco, cassetta, CD)

MUSICDB.BAS fornisce anche un numero di funzioni che caratterizzano lo "stile del database" e che aiutano a controllare al meglio i dati. MUSICDB.BAS permette infatti di:

- Archiviare in modo permanente informazioni sulla collezione in un file sequenziale.
- Esaminare i registri del database uno alla volta sullo schermo.

- Stampare l'intero database della collezione di musica.
- Cercare un registro per titolo o artista.
- Caricare ed esaminare altri database su collezioni di musica.
- Uscire temporaneamente in DOS per eseguirne un comando.

Buona parte del codice di MUSICDB.BAS è dedicato alla creazione di una interfaccia basata su menu per darvi un'idea di come vengono create le schermate di applicativi di funzionalità generale. In questo senso MUSICDB.BAS è abbastanza generico da poter essere facilmente modificato per la raccolta di altri tipi di informazioni.

Registrare informazioni con un database

Un database è una raccolta di registri individuali caratterizzati da un comune formato. l'elenco del telefono, per esempio, è un database: contiene un elenco alfabetico di registri di nomi, ciascuno dei quali contiene elementi quali l'indirizzo e il numero di telefono. Un database ha sempre una struttura fissa; ciò significa che ogni registro contiene lo stesso *tipo* di informazioni anche se non lo stesso *contenuto*. I database oggi trovano largo impiego sia in casa che al lavoro; eccovi qualche altro esempio:

- Registri dei dipendenti.
- Registri degli studenti.
- Prodotti disponibili.
- Collezioni musicali.
- Collezioni di monete.
- Collezioni di film e video.
- Cataloghi di prodotti.
- Cataloghi di biblioteche.
- Date da ricordare.
- Inventari.
- Statistiche sportive.
- Operazioni finanziarie.

Un primo sguardo a MUSICDB.BAS

Prima di scrivere il programma MUSICDB.BAS cerchiamo di capire cosa può fare il nostro programma.
Sullo schermo appariranno gli elementi di seguito riportati.

- *Nome del file corrente* è il nome del file di database musicale attualmente aperto (MUSIC.DB è il file predefinito).
- *Modalità corrente* è il compito attualmente in esecuzione; questo indicatore cambia quando cambiano i compiti eseguiti.
- *Ora corrente* è l'ora segnata dall'orologio di sistema (aggiornata ogni volta che il menu principale viene visualizzato).
- *Menu principale* contiene le sette diverse funzioni che il programma può svolgere; questo è il menu principale che ricompare terminata l'esecuzione di ogni compito.
- *Linea di stato* contiene istruzioni e informazioni sulla selezione di menu attiva.

Prendiamo ora in esame le funzioni contenute nel menu principale per capire come possono essere usate.

L'opzione AGGIUNGI

L'opzione Aggiungi serve ad aggiungere elementi al database. Per selezionare Aggiungi basta digitare 1 e premere Invio; MUSICDB cambia la modalità corrente in AGGIUNGI e richiede le informazioni sulla vostra collezione. La finestra di immissione dovrebbe apparire più o meno in questo modo:

```
Titolo: Lo Schiaccianoci
Artista: Orchestra Filarmonica di Vienna
Anno di pubblicazione: 1982
Stile musicale: Classica
Supporto: Disco

Titolo: Polka norvegese
Artista: Gruppo di polka norvegese
Anno di rilascio: 1953
Stile musicale: Polka
Supporto: Disco
```

```
Titolo: Memorie del passato
Artista: I ricordi
Anno di rilascio: 1964
Stile musicale: Popolare
Supporto: Cassetta

Titolo: Hurried honeymoon
Artista: The Magees
Anno di rilascio: 1988
Stile musicale: Country
Supporto: Compact disc

Titolo: L'anello dei Nibelunghi
Artista: Orchestra filarmonica di Vienna
Anno di rilascio: 1985
Stile musicale: Classica
Supporto: Cassetta

Titolo: FINE
```

Se inserite un certo numero di elementi la finestra inferiore scorre, mentre restano al loro posto la parte inferiore e superiore (grazie all'enunciato VIEW PRINT). Quando digitate FINE il programma trasferisce le informazioni in un file sequenziale su disco e ritorna al menu principale.

L'opzione MOSTRA

L'opzione MOSTRA permette di esaminare il database un registro alla volta. Per selezionare MOSTRA basta digitare *2* e premere Invio. Se continuate a premere Invio incontrate il messaggio di raggiungimento della fine del file che indica che tutti i registri sono stati visualizzati. Notate che ogni registro appare sullo schermo esattamente come era stato inserito. Se premete ancora Invio tornate al menu principale.

L'opzione PRINT

L'opzione PRINT permette di stampare il database. Per selezionarla basta digitare *3* e premere Invio. Se avete una stampante collegata al sistema assicuratevi che sia accesa e pronta per la stampa e solo allora premete *S* per inviarvi i contenuti del file corrente. Se non avete una stampante collegata

non premete *S* altrimenti il programma comincia un'inutile ricerca della stampante; premete invece *R* per tornare al menu principale.

L'opzione CERCA

L'opzione CERCA permette di cercare un titolo o un artista specifico nel database. Per selezionare questa a opzione basta digitare *4* e premere Invio. Alla richiesta della categoria cercata digitate *1* per cercare un titolo o *2* per cercare un artista. Osservate cosa accade se avete inserito i registri del nostro esempio e avete cercato le opere della Orchestra Filarmonica di Vienna.

Due opere vengono visualizzate; notate che l'elemento cercato (in questo caso l'artista) viene evidenziato in verde. Completata la ricerca premete Invio per tornare al menu principale.

L'opzione CAMBIA

L'opzione CAMBIA permette di cambiare il nome di un file con cui si sta lavorando rendendo possibile l'esame o la creazione di un altro database musicale (nello stesso formato). Per selezionare CAMBIA basta digitare *5* e premere Invio. Per permettere di vedere i file disponibili, CAMBIA mostra il contenuto della directory corrente prima di chiedere il nome del nuovo file. Anche in questo caso è possibile includere un'unità o un percorso particolare, ma se il nome di file indicato non è valido il programma viene interrotto. Se si preme Invio senza specificare un nuovo nome di file, il programma seleziona il database predefinito, MUSIC.DB.

NOTA: Col comando CAMBIA specificate solo dei file di database musicali (nuovi o esistenti); non indicate dei file di documenti o di programmi di altre applicazioni (quali quelli con estensione .BAS, .BAT o .DOC) perché potreste danneggiarli.

L'opzione ESCI

L'opzione ESCI permette di uscire temporaneamente in DOS per eseguire dei comandi o piccoli programmi (non avrete la possibilità di eseguire programmi di grosse dimensioni perché QBasic e MUSICDB continuano a essere presenti in memoria). Per selezionare questa opzione si digita *6* e poi si preme

Invio. Dopo aver eseguito i comandi di DOS desiderati basta digitare *exit* e premere Invio per tornare nel menu principale di DOS.

L'opzione FINE

L'opzione FINE permette di uscire dal programma e tornare in QBasic; per selezionare questa opzione basta digitare *7* e premere Invio.

Il listato del programma MUSICDB.BAS

Dopo aver visto come funziona il programma, diamo un'occhiata al listato per vedere come vengono usati gli enunciati e le funzioni appresi fino a questo momento. MUSICDB.BAS (figura 10-14) consiste di un programma principale e di sette sottoprogrammi; ciascuno di questi, anche se gestisce un compito specifico, è disegnato per essere una routine di carattere generale così da poter essere usata in programmi di database di altro genere.

Alcuni degli elementi nel listato sono evidenziati in nero per indicare i punti in cui il programma dev'essere modificato se si decide di adattare il database per registrare altri tipi di informazioni.

Esaminate il listato leggendo i commenti di spiegazione; potrete notare le parti di ogni sottoprogramma che sono indipendenti dal tipo di dati con cui si lavora: la scelta di un elemento dal menu principale, l'elaborazione della selezione effettuata, l'apertura e la chiusura dei file, la lettura, la stampa e la ricerca degli oggetti. Ciò significa che per modificare il programma e adattarlo ad altri tipi di database (raccolta di monete, registro dei dipendenti etc.) basta rimuovere i campi e le richieste di input specifici, sostituendoli con campi e richieste appropriate al nuovo database. L'aspetto della finestra cambierà, ma la struttura interna rimarrà invariata.

Esercizio: Creazione del programma per la collezione musicale

Scrivete ed eseguite il programma MUSICDB.BAS. Notate che la figura 10-15 mostra il programma come apparirebbe se venisse stampato e non come compare all'interno di QBasic. Per scrivere questo programma dovete usare il comando Nuovo SUB dal menu Modifica per inserire ciascun sottoprogramma in una finestra separata (per maggiori informazioni su questa procedura si veda il capitolo 7).


```
' MUSICDB.BAS
' Questo è un semplice programma di database che registra una raccolta
di dati relativi a una collezione di supporti musicali (dischi, cassette
ecc.)
' con un file sequenziale e un gruppo di sottoprogrammi generali.

DECLARE SUB MostraIntestaz ()   ' dichiara i sottoprogrammi
DECLARE SUB SelezioneDiMenu (scelta%)
DECLARE SUB AggiungiRegistri ()
DECLARE SUB MostraRegistri ()
DECLARE SUB StampaRegistri ()
DECLARE SUB Cerca ()
DECLARE SUB CambiaNomeFile ()

COMMON SHARED nomefile$, tmp$   ' dichiara le variabili globali
nomefile$ = "MUSIC.DB"' nome del file di database predefinito
OPEN nomefile$ FOR APPEND AS #1: CLOSE #1   ' si assicura che il file
                                            ' esista
tmp$ = "Anno:   ####  Stile:  \ \  Supporto:  \ \"

MostraIntestaz  ' richiama il sottoprogramma che imposta lo schermo

DO
   SelezioneDiMenu scelta%  ' richiama il sottoprogramma che
                 ' individua il menu scelto

   SELECT CASE scelta%   ' elabora il menu scelto
      CASE 1' "1" significa aggiungi al database
         LOCATE 3, 47: PRINT "AGGIUNGI " ' cambia la modalità in AGGIUNGI
         AggiungiRegistri   ' richiama il sottoprogramma per
                            ' l'aggiunta di elementi
      CASE 2' "2" significa mostra il database
         LOCATE 3, 47: PRINT "MOSTRA "  ' cambia la modalità in MOSTRA
         MostraRegistri  ' richiama il sottoprogramma per mostrare
                         ' gli elementi
      CASE 3' "3" significa stampa il database
         LOCATE 3, 47: PRINT "STAMPA "  ' cambia la modalità in STAMPA
         StampaRegistri  ' richiama il sottoprogramma per la stampa
                         ' di elementi
      CASE 4' "4" significa cerca il database
         LOCATE 3, 47: PRINT "CERCA " ' cambia la modalità in CERCA
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*

```
        Cerca ' Richiama il sottoprogramma per la ricerca di elementi
      CASE 5' "5" significa cambia il nome del file
        LOCATE 3, 47: PRINT "CAMBIA "  ' cambia la modalità in CAMBIA
        CambiaNomeFile ' richiama il sottoprogramma per cambiare nome
                       ' ai file
      CASE 6' "6" significa esci in DOS
        CLS' pulisce lo schermo
        SHELL ' esce nella shell di DOS
        MostraIntestaz ' imposta lo schermo al rientro
      CASE 7' "7" significa fine del programma
        LOCATE 3, 47: PRINT "FINE " ' cambia la modalità in FINE
    END SELECT

LOOP UNTIL (scelta% = 7) ' Ripete il loop fino a quando si sceglie FINE

END

SUB AggiungiRegistri

' Questo sottoprogramma aggiunge nuovi voci musicali al database

LOCATE 25, 1 ' stampa un messaggio nella barra di stato
PRINT "Inserite i dati musicali. Digitate FINE come titolo per uscire."
VIEW PRINT 5 TO 23 ' attiva una finestra di testo (linee 5-23)
PRINT ' richiede le informazioni
PRINT "Inserite le nuove informazioni musicali (senza l'uso di virgole)"
PRINT

OPEN nomefile$ FOR APPEND AS #1' apre il database in modalità append

' fornisce i registri per il file fino a che l'utente digita FINE come
titolo.

WHILE (UCASE$(titolo$) <> "FINE")
   INPUT "Titolo: ", titolo$' riceve l'elemento titolo...
   IF (UCASE$ (titolo$) <> "FINE") THEN
      INPUT "  Artista: ", artista$  '...e le altre informazioni
      INPUT "  Anno di pubblicazione: ", anno%
      INPUT "  Stile musicale: ", stile$
      INPUT "  Supporto: ", supporto$
      PRINT
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*


```
   ' trasferisce i registri nel file del database
      WRITE #1, titolo$, artista$, anno%, stile$, supporto$
   END IF
WEND

CLOSE #1 ' chiude il file quando si è terminato

END SUB

SUB CambiaNomeFile

' Questo sottoprogramma cambia il nome del file di database
' attuale. Se il nuovo file non esiste, il programma lo crea.
' Se non viene specificato alcun nome, il programma usa il file
' predefinito
' MUSIC.DB. Nota: Questo sottoprogramma non controlla che il nome sia
' valido sotto DOS (se si specifica un nome non valido il programma
' viene automaticamente interrotto).

LOCATE 25, 1: PRINT "Specificate il nuovo nome del database musicale...";
VIEW PRINT 5 TO 23 ' Stampa il messaggio nella barra di stato

PRINT ' richiede il nuovo nome del file
PRINT "Usa quest'opzione per creare un nuovo file di database";
PRINT "musicale o per aprirne uno esistente."
PRINT
PRINT "La directory corrente contiene i seguenti file:"
PRINT
FILES "*.*" ' mostra tutti file nella directory corrente
PRINT
PRINT "Con quale file di database musicale vuoi lavorare?"
PRINT "(Premete Invio per il file predefinito MUSIC.DB)"
PRINT
INPUT "Nome del file: ", nomefile$ ' Assegna l'input a una variabile
globale

IF (nomefile$ = "") THEN ' se non viene specificato alcun nome...
   nomefile$ = "MUSIC.DB"' imposta quello predefinito, MUSIC.DB
ELSE  ' altrimenti toglie gli spazi vuoti alle
   nomefile$ = LTRIM$ (RTRIM$(UCASE$(nomefile$)))
END IF' estremità del nome del file e lo rende maiuscolo
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*

```
OPEN nomefile$ FOR APPEND AS #1 ' apre e chiude il file per assicurarsi
CLOSE #1  ' che sia presente nel disco (questo per evitare
   ' errori durante l'apertura in modalità INPUT)
END SUB

SUB MostraIntestaz

' Questo sottoprogramma mostra le informazioni sul programma nelle
' prime tre righe dello schermo e le due righe di divisione che separano
' la finestra contenente queste informazioni sul programma.

CLS' pulisce lo schermo

COLOR 9   ' imposta il colore celeste

PRINT "  COLLEZIONE DI MUSICA"
PRINT
PRINT "File: "; ' mostra i campi per le informazioni sul programma
PRINT "Modalità:   ";
PRINT "Ora:"

PRINT STRING$(80, "-")' stampa le linee di divisione
LOCATE 24, 1: PRINT STRING$(80, "-"); ' nelle righe 4 e 24

COLOR 7   ' imposta il colore bianco predefinito

END SUB

SUB SelezioneDiMenu (scelta%)

' Questo sottoprogramma riceve la scelta di un menu dall'utente
' e la restituisce al programma principale nella variabile scelta%.
' L'enunciato VIEW PRINT viene usato per attivare e disattivare
' la finestra di testo (righe 5-23). Le informazioni visualizzate in
' quest'area
' non disturbano i dati nelle righe tra 1 e 4 e tra 24 e 25.

scelta% + 0  ' inizializza scelta% partendo da 0

VIEW PRINT   ' disattiva la finestra di testo per aggiornare le righe
             ' da 3 a 25
LOCATE 3, 16: PRINT "  ": LOCATE 3, 16: PRINT nomefile$
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*


```
LOCATE 3, 47: PRINT "SELECT"' imposta la modalità select
LOCATE 3, 76 PRINT LEFT$(TIME$, 5) ' aggiorna l'ora corrente
LOCATE 25, 1: PRINT "Digitate un numero tra 1 e 7 e premete Invio...";
VIEW PRINT 5 TO 23 ' attiva la finestra di testo (righe 5-23)
CLS 2 ' pulisce la finestra di testo per le opzioni

PRINT ' richiede all'utente di effettuare una scelta
PRINT "Selezionate un'opzione:"
PRINT
PRINT " 1) AGGIUNGI dati al database musicale e salva su disco"
PRINT " 2) MOSTRA i contenuti del database musicale sullo schermo"
PRINT " 3) STAMPA il database musicale"
PRINT " 4) CERCA una specifica voce del database"
PRINT " 5) CAMBIA il nome del file del database musicale"
PRINT " 6) ESCI temporaneamente in DOS (digitare 'exit' per rientrare)"
PRINT " 7) FINE del programma"
PRINT
  ' la scelta dev'essere un intero tra 1 e 7
DO WHILE (scelta% < 1) OR (scelta% > 7)
   INPUT "Scelta (1-7): ", scelta%
LOOP

CLS 2 ' pulisce la finestra di testo per l'opzione effettuata
VIEW PRINT   ' disattiva la finestra di testo per ripulire la barra
             ' di stato
LOCATE 25, 1: PRINT STRING$(80, " "); ' stampa una riga vuota

END SUB

SUB StampaRegistri

' Questo sottoprogramma invia i contenuti del file del database
' corrente alla stampante.

VIEW PRINT 5 TO 23 ' attiva la finestra di testo (righe 5-23)
PRINT ' mostra il messaggio di introduzione
PRINT "Questa opzione invia i contenuti di "; nomefile$;
PRINT "alla stampante."

VIEW PRINT   ' disattiva la finestra di testo per l'aggiornamento
LOCATE 25, 1 ' della linea di stato
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*

```
INPUT ; "Digitate S per stampare o R per tornare al menu principale: ",
risposta$
VIEW PRINT 5 TO 23 ' attiva la finestra di testo (righe 5-23)
    ' se l'utente vuole stampare (S o s)
IF (risposta$ = "S") OR (risposta$ = "s") THEN
    OPEN nomefile$ FOR INPUT AS #1  ' apre il file del database musicale
    ' invia il titolo alla stampante
    LPRINT "-------------------Collezione di Musica--------------------"
    LPRINT
    LPRINT "Data di stampa: "; DATE$   ' stampa la data corrente
    LPRINT "Nome del file: "; nomefile$   ' stampa il nome del file
    LPRINT
    LPRINT "Contenuto della collezione:"
    LPRINT
    ' Fino a quando il contenuti non è esaurito...
    DO WHILE (NOT EOF(1)) ' legge un registro dal file...
       INPUT #1, titolo$, artista$, anno%, stile$, supporto$

       LPRINT "Titolo:  "; titolo$ ' e ne stampa i campi
       LPRINT "Artista:  "; artista$
       LPRINT "Anno:  "; anno%
       LPRINT "Stile:  "; stile$
       LPRINT "Supporto:  "; supporto$
       LPRINT
    LOOP

LPRINT CHR$(12) ' invia alla stampante un carattere di formattazione
CLOSE #1 ' chiude il file
END IF

END SUB

SUB Cerca

' Questo sottoprogramma cerca i registri contenuti nel file che
corrispondono
' a una certa stringa definita dall'utente. Questa funzione "
' supportata solo per i campi titolo e artista (campi ulteriori possono
' essere inclusi aggiungendo ulteriori enunciati CASE).

num% = 0 ' inizializza la variabile di categoria
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*


```
trova% = 0   ' inizializza il flag dei registri trovati

LOCATE 25, 1 ' aggiorna la linea di stato
PRINT "Inserite la categoria e il contenuto della ricerca...";
VIEW PRINT 5 TO 23 ' attiva la finestra di testo (righe 5-23)

PRINT
PRINT "Selezionate una categoria di ricerca:"  ' richiede l'argomento
                                               ' da cercare
PRINT
PRINT " 1) Cerca in base al titolo"
PRINT " 2) Cerca in base all'artista"
' aggiungete eventuali categorie addizionali in questa posizione
PRINT

DO WHILE (num% < 1) OR (num% > 2)  ' riceve il numero associato
   INPUT "Categoria (1-2): ", num% ' all'argomento da cercare
LOOP

PRINT ' richiede la stringa da cercare
INPUT "Scrivete la stringa da cercare: ", cercaStr$
PRINT
PRINT "Risultato della ricerca:"  ' mostra i risultati della ricerca
PRINT

OPEN nomefile$ FOR INPUT AS #1 ' apre il file del database

DO WHILE (NOT EOF(1)) ' legge i registri dal file
   INPUT #1, titolo$, artista$, anno%, stile$, supporto$

   SELECT CASE num%  'usa num% per confrontare il campi del registro
   CASE 1' se num% = 1, determina se la stringa è nel campo titolo
      IF INSTR(UCASE$(titolo$), UCASE$(cercaStr$) THEN
         trova% = -1 ' se lo è, imposta trova come vero
         COLOR 2: PRINT "Titolo: "; titolo$: COLOR 7
         PRINT "Artista: "; artista$
         PRINT USING tmp$; anno%; stile$; supporto$
         PRINT ' mostra i campi con il campo titolo in verde
      END IF
   CASE 2' se num% = 2, determina se la stringa è nel campo artista
      IF INSTR(UCASE$(artista$), UCASE$(cercaStr$) THEN
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*

```
        trova% = -1 ' se lo è, imposta trova come vero
        COLOR 2: PRINT "Titolo: "; titolo$: COLOR 7
        PRINT "Artista: "; artista$
        PRINT USING tmp$; anno%; stile$; supporto$
        PRINT ' mostra i campi con il campo artista in verde
      END IF
    ' aggiungete enunciati CASE per categorie addizionali
    ' in questo punto...
    END SELECT
LOOP

CLOSE #1 ' chiude il file
IF (NOT trova%) THEN ' se il file non viene trovato...
    COLOR 2: PRINT cercaStr$;' stampa il messaggio "non trovato"
    COLOR 7: PRINT " non trovato nel database "; nomefile$
END IF

VIEW PRINT  ' disattiva la finestra di testo e aggiorna la
            ' linea di stato
LOCATE 25, 1: INPUT ; "Premere Invio per tornare al menu principale...",
richiesta$

END SUB

SUB MostraRegistri

' Questo sottoprogramma mostra tutti i registri del database
' sullo schermo, uno alla volta.

LOCATE 25, 1 ' Aggiorna la linea di stato
PRINT "Premere Invio per continuare...";

VIEW PRINT 5 TO 23 ' attiva la finestra di testo (righe 5-23)
PRINT ' mostra il messaggio di apertura
PRINT "Questa opzione permette di vedere la collezione di musica ";
PRINT "un registro alla volta."
PRINT

OPEN nomefile$ FOR INPUT AS #1 ' apre il file del database

DO WHILE (NOT EOF(1)) ' prende il registro dal file
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database (continua).*


```
    INPUT #1, titolo$, artista$, anno%, stile$, supporto$

    PRINT "Titolo:  "; titolo$  ' Mostra ciascun campo sullo schermo
    PRINT "Artista:  "; artista$
    PRINT USING tmp$; anno%; stile$; supporto$

    INPUT " ", richiesta$ ' fa una pausa dopo ogni registro
LOOP

CLOSE #1 ' chiude il file
PRINT "** Fine del file cercato **"   ' mostra il messaggio EOF
                                      ' (fine del file)
INPUT " ", richiesta$ ' fa una pausa prima di tornare
    ' al menu principale
END SUB
```

Figura 10-14. *MUSICDB.BAS, programma database che utilizza un file sequenziale per generare un elenco dei pezzi musicali. Modificare le voci in neretto quando il programma verrà adattato a un altro tipo di database.*

SOMMARIO

In questo capitolo abbiamo iniziato a riunire alcune delle tecniche sviluppate fino ad ora; abbiamo lavorato con grandi quantità di dati, vedendo come si leggono e si scrivono file sequenziali. Abbiamo anche esaminato quale ruolo il DOS e i suoi comandi giocano nella programmazione in QBasic e abbiamo usato molti di questi comandi direttamente. Inoltre abbiamo creato un programma di database per registrare i dati di una collezione di supporti musicali che combina le tecniche per la gestione delle stringhe e dell'interfaccia utente, viste nei capitoli precedenti, con quelle di gestione di file sequenziali, viste in questo capitolo; il risultato è un database di carattere generale che può essere usato per registrare informazioni di qualunque genere.

DOMANDE ED ESERCIZI

1. Quale differenza c'è tra l'apertura di un file in modalità OUTPUT e in modalità APPEND?

2. Quale dei seguenti enunciati racchiude le informazioni tra virgolette quando le invia a un file sequenziale?

a. INPUT#
b. PRINT# USING
c. PRINT#
d. WRITE#

3. Vero o Falso: il nome di file specificato in un'enunciato OPEN dev'essere in maiuscolo.
4. Quando si può dire che l'enunciato LINE INPUT# sia più utile dell'enunciato INPUT#?
5. Che errore c'è in quest'enunciato di QBasic?

```
SHELL COPY TEST.TXT TEST2.TXT
```

6. Che cosa mostrano queste righe di programma?

```
cercaStr$ = "Yellow Submarine"
artista$ = "The Beatles"
IF INSTR(UCASE$(artista$), UCASE$(cercaStr$)) THEN
  PRINT "Stringa trovata!"
ELSE
  PRINT "Stringa non trovata"
END IF
```

7. Scrivete un programma che richiede all'utente un elenco di città e le archivia in un file sequenziale. Progettate il programma in modo che l'utente possa vedere i contenuti del file dopo averlo creato.
8. Scrivete un programma che richiede all'utente un elenco di nomi e indirizzi, li archivia in un file sequenziale e poi ordina i registri alfabeticamente per nome, in un file a parte. Suggerimento: il modo più semplice per risolvere questo problema è quello di usare una matrice per conservare e ordinare i registri. Vi consigliamo di usare il sottoprogramma Shell Sort visto nel capitolo 9.

CAPITOLO 11

Lavorare con grafica e suoni

Dopo aver visto i fondamenti di QBasic possiamo analizzare come è possibile migliorare i programmi aggiungendo grafica e suoni. In questo capitolo vedremo quali tipi di grafica il vostro computer è in grado di creare e come è possibile usare il suono per vivacizzare i vostri programmi.

INTRODUZIONE ALLA PROGRAMMAZIONE GRAFICA

L'hardware determina il tipo di grafici che è possibile creare ed è perciò importante sapere che tipo di apparecchiatura video è installata sul computer.

L'apparecchiatura video

In un computer IBM compatibile l'apparecchiatura video è composta da due elementi:

- L'*adattatore video*, cioè la scheda elettronica installata dentro la macchina (Nota: nei computer IBM PS/2 o compatibili questo elemento è parte del computer, non è cioè una scheda aggiunta).
- Il *monitor* (detto anche *schermo*).

Alcune apparecchiature video sono in grado di visualizzare sia la grafica sia il colore, altre solo la grafica e altre ancora non sono in grado di mostrare né la grafica né il colore. Inoltre tra i sistemi in grado di mostrare sia la grafica che il colore c'è un'ampia varietà di scelta. Generalmente meno costosa è l'apparecchiatura, meno colori saranno disponibili e più sgranata risulterà l'immagine; un'apparecchiatura video costosa è spesso in grado di visualizzare molti colori e di produrre un'immagine molto più professionale.

Adattatori video

La tabella 11-1 elenca gli adattatori video disponibili per computer IBM PC, IBM PC/XT, IBM PC/AT e compatibili.

Tabella 11-1. *Adattatori video per i computer PC IBM e compatibili.*

Adattatore	Video multicolore	Grafica
MDA (monochrome display adapter)	No	No
HGC (Hercules graphics card)*	No	Si
CGA (color/graphics adapter)	Si	Si
EGA (enhanced graphics adapter)	Si	Si
MCGA (multicolor gate array)	Si	Si
VGA (video graphics array)	Si **	Si

* *La scheda grafica Hercules non è prodotta da IBM. Questa scheda viene spesso indicata come adattatore monografico o adattatore grafico monocromatico.*
** *Per avere più colori è necessario disporre di un monitor VGA.*

È possibile installare diversi tipi di adattatori video, ma la maggior parte sono compatibili con quelli elencati nella precedente tabella. Se il vostro computer dispone solo di una scheda MDA non sarete in grado di svolgere la maggior parte degli esercizi mostrati nel capitolo. Se non siete sicuri del tipo di adattatore di cui disponete proseguite con la lettura; fra un po' vedremo un programma in grado di dirvi se potrete svolgere o meno gli esercizi successivi.

Monitor

Anche i monitor sono disponibili sul mercato in grande varietà (di tipi e di prezzi). Possiamo notare che, sebbene certi adattatori video funzionino con molti diversi tipi di monitor, tuttavia i produttori cercano generalmente di abbinare l'adattatore col monitor più appropriato.

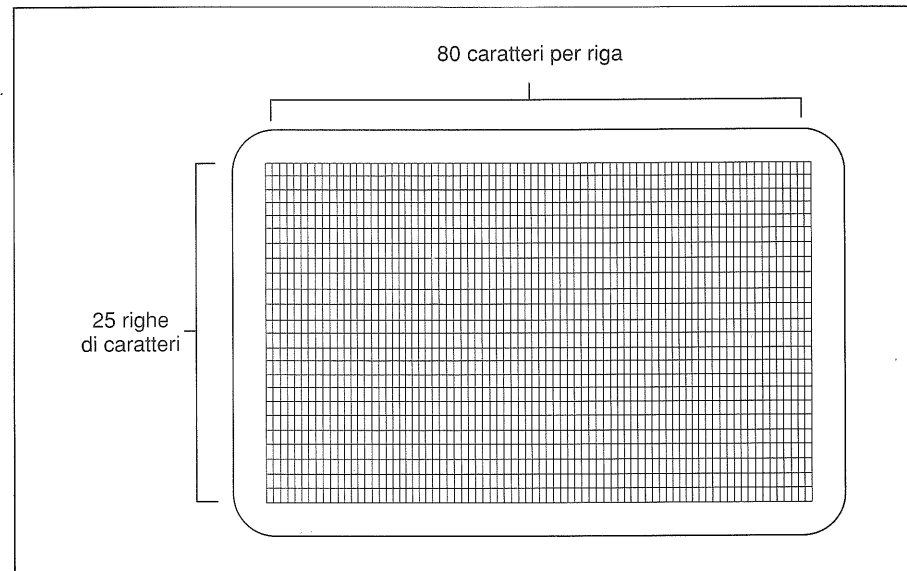
Modalità testo e modalità grafica

Tutti gli adattatori capaci di visualizzare grafica possono operare in *modalità testo* o in *modalità grafica*.

- In modalità testo l'adattatore video può visualizzare caratteri alfanumerici (lettere, numeri e segni di punteggiatura) ma non forme (righe, riquadri, cerchi etc.).
- In modalità grafica l'adattatore video può visualizzare caratteri alfanumerici ma anche forme non alfanumeriche, quali cerchi o poligoni.

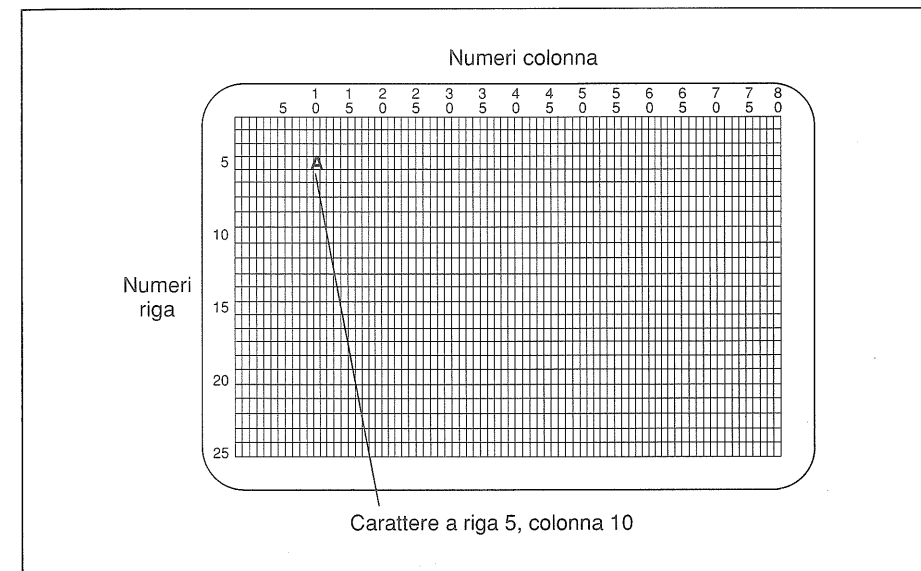
  Cercheremo di analizzare queste due modalità in maggior dettaglio.

MODALITÀ TESTO

All'avvio di QBasic l'adattatore video mostra sia la Finestra di digitazione , che lo schermo di output in modalità testo: 25 righe con 80 caratteri per riga.

Ogni riga e colonna nello schermo di output è numerata e a ogni riquadro vengono attribuite delle coordinate uniche, dette coordinate dello schermo. Ognuna di queste caselle può contenere solo un carattere alfanumerico.

Il cursore di testo

Ricorderete che quando digitate un programma nella Finestra di digitazione il cursore lampeggiante indica dove apparirà il carattere successivo. Nello schermo di output QBasic usa il proprio cursore, detto anche cursore di testo, per determinare dove appariranno i caratteri. A differenza del cursore lampeggiante della Finestra di digitazione, il cursore di testo è invisibile.

L'enunciato CLS, che pulisce lo schermo, svolge in realtà anche un'altra funzione: serve a impostare il cursore di testo in corrispondenza della prima riga e della prima colonna dello schermo di output; quando QBasic esegue un enunciato PRINT dopo CLS, ne stampa il messaggio nella prima riga a partire dalla prima colonna. Supponiamo ora che PRINT non termini con un punto

e virgola, QBasic allora sposta il cursore di testo in corrispondenza della seconda riga e della prima colonna.

L'enunciato LOCATE

È possibile modificare il posizionamento automatico del cursore di testo con l'enunciato LOCATE che permette di stabilire dove deve apparire il cursore di testo all'interno di un programma.

L'enunciato LOCATE presenta la seguente sintassi:

```
LOCATE [riga][, colonna]
```

riga è un intero da 1 a 25; *colonna* è un intero da 1 a 80.

- Se omettete l'indicazione *riga* il cursore di testo rimane in corrispondenza della riga corrente.
- Se omettete l'indicazione *colonna* il cursore di testo rimane in corrispondenza della colonna corrente.
- Se omettete entrambe le indicazioni QBasic lascia il cursore di testo nella posizione corrente.
- Se per *riga* o *colonna* specificate un valore al di fuori delle dimensioni dello schermo, otterrete un messaggio di errore.

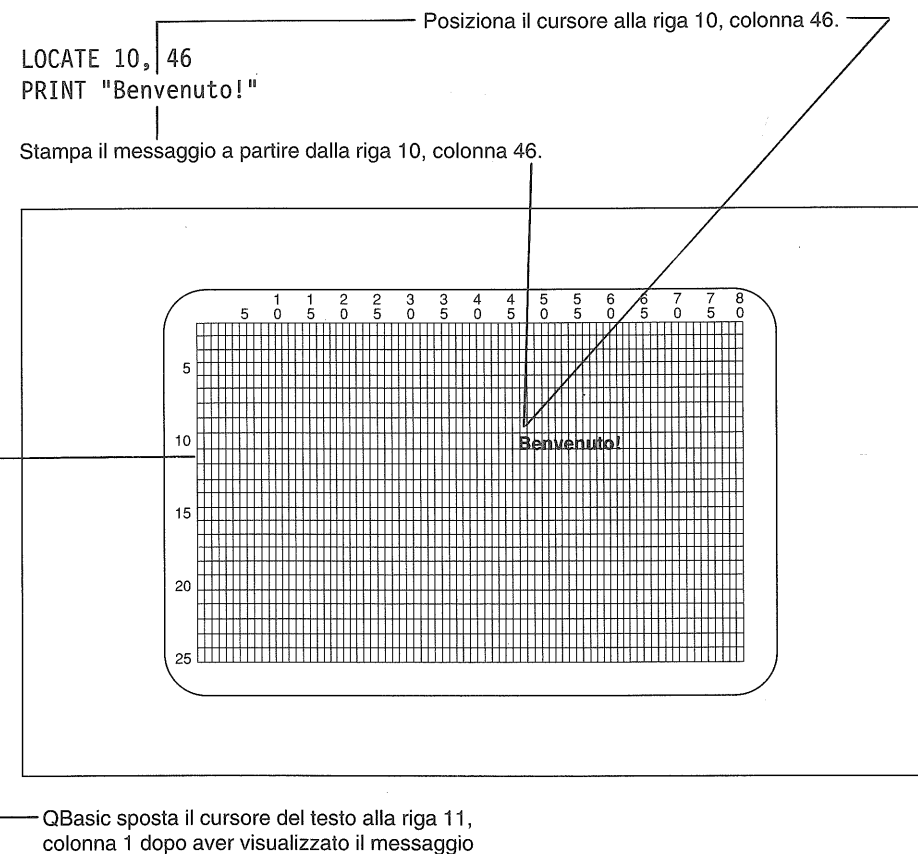
NOTA: Come vedremo fra poco, è possibile far in modo che l'adattatore video mostri solo 40 colonne. In questo caso la colonna non può avere un valore maggiore di 40.

Nell'esempio che segue, LOCATE indica a QBasic di posizionare il cursore alla riga 10 e alla colonna 46. Quando QBasic esegue il successivo enunciato PRINT stampa il messaggio *Benvenuto!* iniziando dalla riga 10 e dalla colonna 46. Dopo aver stampato il messaggio, QBasic posiziona il cursore alla riga 11 e alla colonna 1 e stampa il messaggio dell'enunciato PRINT successivo cominciando in quel punto, a meno che vi sia un altro enunciato LOCATE che cambia la posizione.

Posiziona il cursore alla riga 10, colonna 46.

```
LOCATE 10, 46
PRINT "Benvenuto!"
```

Stampa il messaggio a partire dalla riga 10, colonna 46.

QBasic sposta il cursore del testo alla riga 11, colonna 1 dopo aver visualizzato il messaggio

Esercizio: Uso dell'enunciato LOCATE

1. Scrivete il programma LOCATE-1.BAS (figura 11-1).

```
' LOCATE.BAS
' Questo programma mostra l'uso dell'enunciato LOCATE
CLS
INPUT "Inserite le coordinate di riga (1-24): ", rigaNum%
PRINT
INPUT "Inserite le coordinate di colonna (1-80): ",colNum%
PRINT
INPUT "Scrivete il messaggio da visualizzare: ", messaggio$

CLS

FOR i% = 0 TO 70 STEP 10 ' stampa le coordinate di colonna
      PRINT "1234567890";       ' in cima allo schermo
NEXT i%

FOR i% = 2 TO 23   ' stampa i numeri di riga lungo
      LOCATE i%, 1 ' il lato sinistro dello schermo
      PRINT LTRIM$(STR$(i%))   ' non stampa uno spazio vuoto
NEXT i%      ' prima di ogni numero

LOCATE rigaNum%, colNum%        ' stampa il messaggio alle
PRINT messaggio$                ' coordinate fornite dall'utente
```

Figura 11-1. *Un programma che mostra l'uso dell'enunciato LOCATE.*

2. Eseguite il programma e specificate valori di riga e colonna maggiori di 2 (ne capirete fra poco la ragione). Dopo aver eseguito il programma lo schermo di output dovrebbe essere simile alla figura di pagina seguente. Notate che il programma stampa i numeri nella riga 1 e nelle colonne 1 e 2; questo vi permette di verificare facilmente che il messaggio cominci effettivamente alle coordinate specificate.
   L'esempio della figura impiega un valore di riga pari a 5, un valore di colonna pari a 20 e stampa il messaggio *Salve!* Eseguite il programma un paio di volte per capire meglio come funziona l'enunciato LOCATE.

La funzione STR$

La funzione STR$ converte un valore numerico in una stringa di numerali. La sintassi di questa funzione è la seguente:

```
STR$(valore)
```

valore è una qualsiasi espressione numerica.

STR$ può essere usato insieme alla funzione LTRIM$ per eliminare gli spazi che compaiono alla sinistra di numeri non negativi. Ciò è utile quando si posiziona un numero con l'enunciato LOCATE; per esempio, gli enunciati successive mostrano il numero 12.34 senza spazi davanti:

```
numero! = 12.34
PRINT LTRIM$(STR$(numero!))
```

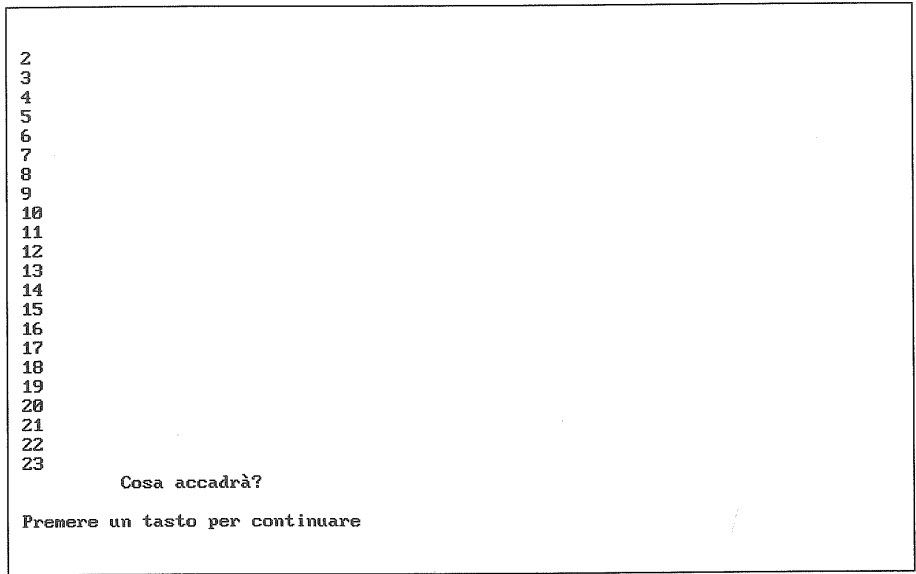
3. Provate ora a eseguire il programma indicando un valore alto di colonna e usando un messaggio abbastanza lungo. Per esempio provate con riga *5*, colonna *70* e col messaggio *Dove comparirà questo messaggio?* Lo schermo di output dovrebbe apparire in modo simile a quello della figura qui riportata.

Notate che QBasic mostra il messaggio cominciando dalla riga 6 e dalla colonna 1, anche se avevate indicato la riga 5. Poiché il messaggio supera il limite degli 80 caratteri, QBasic sa che non può stare interamente nella riga 5 e perciò lo fa cominciare dalla riga successiva. Tenete presente questo comportamento di QBasic quando usate LOCATE e agite di conseguenza.

Come potete vedere il programma chiede di indicare un valore di riga da 1 a 23 e le righe sullo schermo sono numerate fino a 23. Prima avevamo però detto il valore di riga può andare da 1 a 25. Perché abbiamo limitato il programma a 23 righe?

4. Eseguite il programma di nuovo e indicate 24 come valore di riga. Il vostro schermo di output apparirà come nella figura riportata a pagina seguente.

   Dopo aver stampato il messaggio, QBasic ha spostato il cursore nella prima colonna della riga successiva; per farlo ha dovuto spostare di una riga verso l'alto i contenuti dello schermo facendo scomparire il contenuto della prima riga.

   Potete risolvere questo problema aggiungendo un punto e virgola alla fine dell'enunciato PRINT, tendendo così il cursore nella stessa riga. Gli enunciati che seguono, per esempio, visualizzano il messaggio *Questa è*

la riga 24 sulla riga 24 senza spostare verso l'alto i contenuti dello schermo:

```
LOCATE 24, 1
PRINT "Questa è la riga 24"
```

Potete usare la stessa tecnica per posizionare del testo nella riga 25 (abbiamo visto qualcosa del genere nel programma MUSICDB.BAS nel precedente capitolo). Ricordate in ogni caso che QBasic usa la riga 25 per mostrare il messaggio *Press any key to continue* (Premere un tasto qualsiasi per continuare) per cui i caratteri collocati su questa riga verranno sovrascritti.

Uso di LOCATE per creare animazioni

L'enunciato LOCATE è utile anche per creare semplici animazioni (dando cioè all'utente *l'illusione* del movimento sullo schermo). Un modo per far ciò è quello di visualizzare un carattere ripetutamente nella direzione del movimento, sostituendo di volta in volta il carattere precedente con uno spazio, come si vede nella figura 11-2. Così facendo, si ottiene l'illusione del movimento.

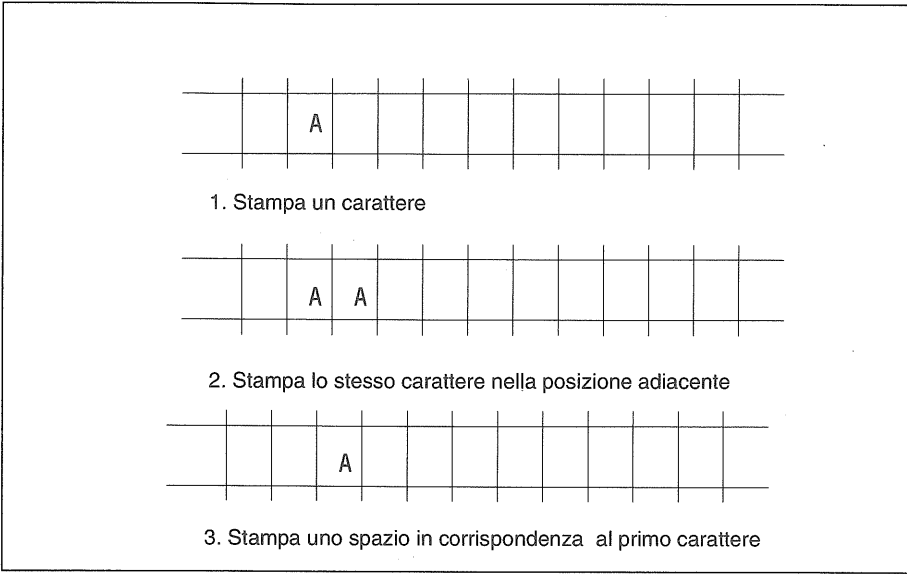



Figura 11-2. *Animazione di un carattere sullo schermo.*

Esercizio: Animazioni basate sul testo

Supponiamo di voler scrivere un programma che sposta la lettera X attraverso lo schermo.

1. Scrivete ed eseguite il programma LOCATE-2.BAS (figura 11-3).

```
' LOCATE-2.BAS
' Questo programma mostra il primo passo nella creazione
' di un'animazione.

CLS

FOR i% = 1 TO 80
     LOCATE 10, i%
     PRINT "X"
NEXT i%
```

Figura 11-3. *Il primo passo nell'animazione di un carattere.*

2. Eseguite il programma; apparirà una serie di X nella riga 10 del vostro schermo. Anche se ciò non dà l'illusione del movimento rappresenta però il primo passo per "animare" un carattere. Siccome QBasic esegue il programma velocemente le X vengono stampate quasi simultaneamente, ma si possono usare le tecniche viste nel capitolo 6 per ritardare il loop nel programma, rallentando la velocità con cui QBasic stampa le X (il prossimo programma include un ritardo nel loop; prima di leggere questo esempio cercate di scriverne uno da soli).
   Il passo successivo è quello di creare l'illusione che una sola lettera X si sposti sullo schermo, anche se il programma stamperà in realtà ottanta X separate. Dunque il programma deve stampare un X, stamparne una seconda immediatamente alla destra della prima e poi cancellare la prima X, inserendo uno spazio al suo posto.

3. Scrivete ed eseguite il programma LOCATE-3.BAS (figura 11-4).

```
' LOCATE-3.BAS
' Questo programma mostra un'animazione semplice da realizzare

CLS

LOCATE 10, 1      ' posiziona la prima X
PRINT "X"
FOR i% = 2 TO 80 ' entra in un loop per il resto del programma
     LOCATE 10, i%
     PRINT "X"   ' stampa una X vicino alla precedente
     LOCATE 10, i% -1
     PRINT " "   ' "cancella" la X precedente
     FOR j% = 1 TO 300 ' ritarda il loop
     NEXT j%
NEXT i%
```

Figura 11-4. *Un programma che anima un carattere.*

Notate che la prima X viene stampata prima che il loop inizi. La ragione va trovata nel modo in cui le X vengono cancellate. I valori di LOCATE per la stampa sono 10, *i%* e *–1*; se il loop iniziasse col valore *1*, il programma non riuscirebbe a stampare il primo spazio perché dovrebbe stamparlo alla riga 10 e alla colonna 0. Poiché 0 non è un valore valido QBasic si interromperebbe e mostrerebbe un messaggio di errore.

4. Eseguite il programma e notate come questo dia effettivamente l'impressione che la lettera X si "sposti" attraverso lo schermo. Se volete, fermatevi a questo punto e fate delle prove con il programma. Vi accorgerete che inserire questo tipo di animazione nei vostri programmi può davvero "catturare" l'attenzione dell'utente.

Animazione verticale

Ora che avete imparato i principi fondamentali dell'animazione del testo, proviamo qualcosa di più divertente: delle parole che scendono dal cielo.

Esercizio: Il programma CADUTA.BAS

Scrivete ed eseguite il programma CADUTA.BAS (figura 11-5).

```
' CADUTA.BAS
' Questo programma mostra un esempio di animazione verticale

CLS

FOR i% = 1 TO 4
     READ colonna%, parola$
     FOR riga% = 2 TO 18
          LOCATE riga%, colonna%
          PRINT parola$
          LOCATE riga% -1, colonna%
          PRINT SPACE$(LEN(parola$))
          SOUND (2400 / riga%), 1
     NEXT riga%
NEXT i%
DATA 28, "Il", 31, "cielo", 37, "sta", 41, "cadendo!"
```

Figura 11-5. *Un programma che mostra l'animazione verticale.*

Invece di stampare solo una lettera, come nell'esempio precedente, questo programma stampa un'intera parola e per "cancellare" quella precedente usa una stringa di spazi.

Inoltre questo programma usa il suono per rafforzare l'impressione della caduta di ciascuna parola. Si noti che l'enunciato SOUND usa il valore corrente di *riga%* come divisore di *2400*. Come risultato si avrà che all'aumentare del valore di *riga%* il risultato della divisione diminuisce, riducendo la tonalità (analizzeremo l'enunciato SOUND in maggior dettaglio più avanti in questo capitolo).

MODALITÀ GRAFICA

Come abbiamo visto la modalità testo è un modo interessante e relativamente diretto di presentare informazioni sullo schermo; si possono infatti aggiungere "grafici" basati sul testo e semplici animazioni senza grossi sforzi di programmazione.

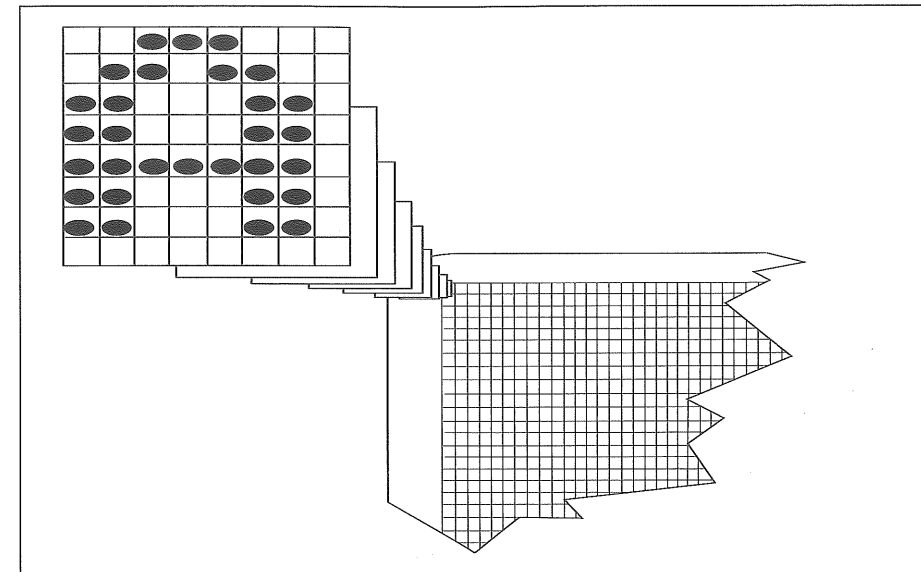
Tuttavia la programmazione basata sul testo presenta dei limiti. Innanzitutto anche il più piccolo carattere (per esempio un punto) copre un'intera casella, con la conseguenza che diventa difficile se non impossibile creare grafici ben definiti. Inoltre, l'animazione basata sul testo non viene visualizzata in modo continuo perché l'oggetto animato deve sempre saltare di casella in casella ogni volta che il programma lo sposta.

Perciò se si vogliono ottenere forme definite o animazioni senza "salti" è consigliabile disporre di un adattatore video che supporta la modalità grafica e usare gli strumenti grafici che QBasic è in grado di offrire.

Cos'è la modalità grafica?

Se guardate attentamente i caratteri sullo schermo noterete che questi sono costituiti di punti piccolissimi che, messi insieme, formano una lettera; infatti ciascuna delle caselle viste in precedenza è sostanzialmente una matrice di 8 per 8 punti, come mostrato nella figura seguente.

Nella figura si vede come la matrice di punti viene riempita per formare la lettera A. Sebbene non tutti i punti vengano utilizzati, tutti i punti della matrice appaiono sullo schermo quando l'adattatore video si trova in modalità testo. È questa la ragione per cui un'intera casella è necessaria per ciascun carattere.

In modalità grafica le regole cambiano perché è possibile avere accesso a tutti i singoli punti.

NOTA: Alcuni adattatori video possono visualizzare matrici di diverse dimensioni. Per esempio, alcuni possono visualizzare matrici di 8 punti in ampiezza e 9 in altezza; altri di 9 punti in ampiezza e 14 in altezza e così via. Per semplicità faremo riferimento sempre a matrici 8 per 8.

Risoluzione grafica

Nel campo della grafica basata su computer, la parola *risoluzione* viene definita come finezza di dettaglio in un'immagine. In linea generale un'immagine a bassa risoluzione presenta un aspetto granulare mentre un'immagine ad alta risoluzione presenta un aspetto continuo, quasi fotografico (maggiore la risoluzione, maggiore la qualità dell'immagine). Se osservate la foto di un giornale da vicino vi accorgerete che è fatta di migliaia di piccolissimi punti, non riconoscibili a distanza.

Anche lo schermo di un computer (così come quello di una televisione) forma le immagini attraverso puntini piccolissimi. Più piccoli sono i punti (cioè maggiore è la risoluzione) migliore sarà la qualità delle immagini.

Sullo schermo di un computer ogni punto viene detto pixel. A differenza dei punti che costituiscono la foto di un giornale (che possono avere dimensioni diverse per aree più chiare o più scure), tutti i pixel dello schermo di un computer hanno la stessa dimensione.

NOTA: Ricordate che è l'adattatore video e non il monitor che determina la risoluzione, così come è responsabile della visualizzazione di caratteri e figure sullo schermo. Perciò quando si parla di hardware e di funzionalità grafiche del computer si fa riferimento sostanzialmente all'adattatore video.

Per poter lavorare in modalità grafica è necessario impostare l'adattatore video su questa modalità. In QBasic ciò viene fatto con l'enunciato SCREEN.

L'enunciato SCREEN

L'enunciato SCREEN imposta l'adattatore video in modalità grafica al momento della visualizzazione del risultato. La forma più semplice di quest'enunciato è la seguente:

```
SCREEN mode
```

mode è un numero da 0 a 13 (esclusi 5 e 6) che dice a QBasic in quale modalità grafica si vuole impostare l'adattatore video. Sebbene vi siano 12 modalità grafiche disponibili il vostro adattatore sarà in grado di usarne solo alcune e potrà operare in una sola modalità alla volta.

Ogni modalità è unica per quanto riguarda la risoluzione e il numero di colori supportati. Alcune modalità possono visualizzare 25 righe e 80 colonne di testo; altre 25 righe e 40 colonne e altre ancora possono visualizzare anche più di 25 righe.

La tabella 11-2 elenca le 14 modalità; le risoluzioni indicate rappresentano la massima risoluzione possibile per quella particolare modalità. I valori sono indicati nella forma *orizzontale * verticale* (perciò, 320*200 significa che lo schermo ha un'ampiezza pari a 320 pixel e un'altezza pari a 200).

Anche il numero di colori rappresenta il massimo disponibile per quella particolare modalità. Si noti che alcuni adattatori potrebbero non essere in grado di visualizzare il numero massimo di colori per una data modalità, anche se l'adattatore video la supporta.

La risoluzione testo di ogni modalità indica la dimensione in cui il testo verrà visualizzato se si usa l'enunciato PRINT. Notate che la maggior parte delle modalità usa 25 righe; ciò che è più importante a questo punto è il numero delle colonne di testo che una data modalità è in grado di visualizzare.

Tabella 11-2. *Modalità schermo disponibili e loro risoluzioni*

Modalità schermo	Risoluzione	Numero di colori	Risoluzione testo
0	(solo testo)	16	80 * 25
1	320 * 200	4	40 * 25
2	640 * 200	2	80 * 25
3*	720 * 348	1	80 * 25
4**	640 * 400	1	80 * 25
5	Non supportato		
6	Non supportato		
7	320 * 200	16	40 * 25
8	640 * 200	16	80 * 25
9	640 * 350	16	80 * 25
10	640 * 350	4***	80 * 25
11	640 * 480	2	80 * 30
12	640 * 480	16	80 * 30
13	320 * 200	256	40 * 25

* *Solo schede grafiche Hercules e compatibili.*
** *Solo personal computer Olivetti e AT&T 6300.*
*** *La modalità 10 visualizza diversi attributi invece di diversi colori. In altre parole le immagini vengono mostrate in una forma regolare, luminosa o lampeggiante invece che in colori diversi.*

Esercizio: Identificazione dell'adattatore del vostro computer

Scrivete ed eseguite il programma TEST-VID.BAS (figura 11-6). Questo programma fornisce un elenco di impostazioni di modalità che possono essere usate con l'enunciato SCREEN.

NOTA: Questo programma usa due nuove enunciati (ON ERROR GOTO e RESUME NEXT) per dire a QBasic cosa fare se incontra una modalità non supportata. Tranne che per questo esempio, ON ERROR GOTO e RESUME NEXT vanno al di là degli obiettivi di questo libro. Per ulteriori informazioni su questi due enunciati consultate la funzione di aiuto in linea di QBasic.

Le coordinate della modalità grafica

Con l'enunciato LOCATE, specificando le coordinate di riga e colonna, si può stabilire dove apparirà il testo sullo schermo di output. In modalità grafi-

```
' TEST-VID.BAS
' Questo programma determina quali modalità sono supportate
' dall'adattatore video

DIM okModi%(18)    ' matrice che contiene le modalità video testate

CLS

ON ERROR GOTO ModoImpossibile  ' dice a QBasic cosa fare nel caso
                               ' in cui trovi un errore

FOR i% = 0 TO 13     ' prova a impostare l'adattatore video
      SCREEN i%      ' in tutte le 14 modalità; se c'è un errore
      IF erroreCodice% = 0 THEN' (cioè se la modalità non è supportata)
            okModi%(i%) = 1    ' passa alla fine del programma
            migliorModo% = i%
      END IF
      erroreCodice% = 0
NEXT i%

SCREEN migliorModo%       ' imposta la migliore modalità per lo schermo
PRINT migliorModo%
FOR i% = 0 TO 13
      IF okModi%(i%) = 1 THEN
            PRINT "puoi usare la modalità"; i%; "col tuo computer"
      END IF
NEXT i%

END
ModoImpossibile:    ' routine di gestione dell'errore
      erroreCodice% = 1  ' (per evitare che QBasic si interrompa
      RESUME NEXT  ' mostrando un messaggio di errore)
```

Figura 11-6. *Un programma che controlla l'adattatore video.*

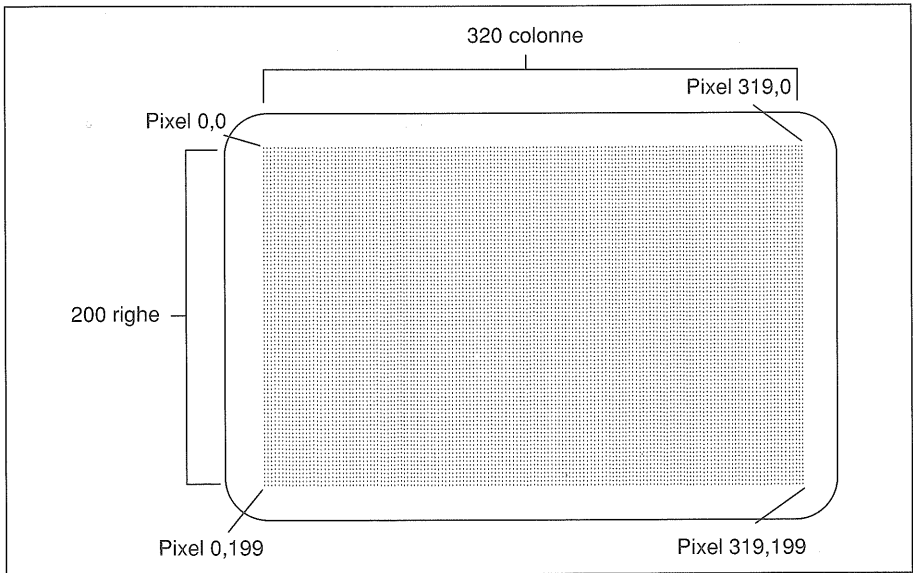
ca lo schermo di output presenta due tipi di coordinate a seconda del tipo di informazione che si vuole visualizzare:

- Coordinate di testo, che funzionano in modo equivalente sia in modalità grafica sia in modalità testo.
- Coordinate di grafica che vengono usate per posizionare sullo schermo informazioni non testuali (come linee o cerchi).

Esaminiamo le coordinate della modalità 1 dello schermo. Come si vede dalla tabella 11-2 la modalità 1 ha una risoluzione grafica di 320*200 e una risoluzione di testo di 40*25, come mostrato nella figura seguente.

Notate una prima importante differenza tra il sistema di coordinate grafiche e di testo: il primo pixel in alto a sinistra sullo schermo grafico ha coordinate *(0,0)* mentre sullo schermo di testo ha coordinate *(1,1)*.

Un'altra importante differenza si ha nel modo in cui le coordinate vengono specificate: in modalità testo dovete indicare *riga e colonna*, in modalità grafica invece, *colonna e riga*.

Si noti anche che in modalità grafica, come anche in modalità testo, ogni parte dello schermo ha coordinate uniche.

Lavorare con singoli pixel

Dopo aver usato l'enunciato SCREEN per impostare l'adattatore video in modalità grafica si possono usare due enunciati di QBasic per collocare pixel individuali in luoghi specifici sullo schermo: PSET e PRESET.

L'enunciato PSET

La sintassi dell'enunciato PSET, nella sua forma più semplice, è la seguente:

`PSET (coordinataX, coordinataY)[, colore]`

coordinataX è il numero della colonna in cui si vuole far apparire il pixel. L'intervallo di valori che si può usare dipende dalla modalità schermo impostata con l'enunciato SCREEN. Per esempio, se si sceglie la modalità 1, il valore di *coordinataX* è compreso tra 0 e 319 perché il numero massimo di colonne in questa modalità è 320.

coordinataY è il numero della riga in cui si vuole far apparire il pixel. Anche in questo caso l'intervallo di valori dipende dalla modalità schermo impostata. Per esempio, se la modalità schermo è 1 il valore di *coordinataY* è compreso tra 0 e 199 perché il numero massimo di righe in questa modalità è 200.

NOTA: Le parentesi attorno ai valori delle coordinate non sono facoltative ma obbligatorie.

colore è un valore che rappresenta il colore del pixel. Anche in questo caso i valori possibili dipendono dalla modalità schermo impostata; se si omette *color* QBasic mostra il pixel nel colore di sfondo corrente.

Un enunciato PSET imposta solo un pixel alla volta, ma può essere posto all'interno di un loop per disegnare più pixel in una riga.

Per cancellare un pixel disegnato con PSET basta porre le coordinate del pixel in un'altro enunciato PSET e specificare il colore dello sfondo come valore di *colore*.

Esercizio: Uso dell'enunciato PSET

1. Scrivete il programma PSET.BAS (figura 11-7).

2. Eseguite il programma e specificate un valore compatibile col vostro adattatore video (ottenuto con l'esercizio precedente). Il programma comincerà a "spostare" un pixel con salti regolari sullo schermo; dopo aver collocato ogni pixel sullo schermo il programma aggiorna il messaggio in fondo allo schermo per mostrare le coordinate del pixel.
   Se col vostro adattatore avete a disposizione più di una modalità, provate a eseguire nuovamente il programma con una diversa modalità. Notate la diversa risoluzione (il programma coprirà quasi completamente di pixel


```
' PSET.BAS
' Questo programma mostra l'uso dell'enunciato PSET

CLS

INPUT "Inserite la modalità schermo desiderata (0-13): ", modoNum%
SCREEN modoNum%

LOCATE 22, 23
PRINT "Col: Riga:"
FOR i% = 0 TO 200 STEP 20' valori per coordinataX
   FOR j% = 0 TO 135 STEP 15' valori per coordinataY
      PSET (i%, j%)   ' usa il colore di sfondo predefinito
      LOCATE 23, 1 ' posiziona il cursore di testo
      PRINT " Coordinate correnti: ("; i%; ","; j%; ")"

      FOR k% = 1 TO 2000 ' ritarda il loop
      NEXT k%

      PSET (i%, j%), 0   ' usa il colore dello sfondo per cancellare
      LOCATE 23, 22
      PRINT SPACE$(13)   ' cancella le coordinate stampate
   NEXT j%
NEXT i%
```

Figura 11-7. *PSET.BAS. Un esempio dell'enunciato PSET.*

uno schermo a bassa risoluzione, per esempio 320*200, mentre coprirà solo l'angolo superiore sinistro di uno schermo ad alta risoluzione, per esempio 640*480.

Modifica dei colori dello sfondo e di quelli in primo piano

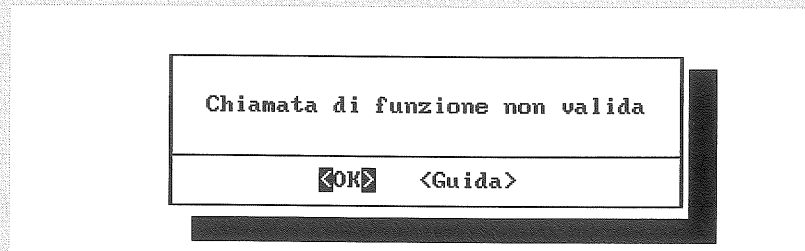
Il capitolo 6 ha introdotto brevemente l'enunciato COLOR, che presenta la seguente sintassi:

`COLOR [primo piano][, sfondo]`

primo piano e *sfondo* sono i valori che rappresentano dei colori in primo piano e sullo sfondo. I valori utilizzabili dipendono dalla modalità grafica in cui si trova l'adattatore video (si veda la tabella 11-2). Se non viene specificato alcun valore, QBasic usa quello corrente; se l'argomento *primo piano* non viene specificato bisogna far precedere l'argomento *sfondo* da una virgo-

Chiamate di funzione non ammesse

Vi capiterà di certo di eseguire un programma che include enunciati grafici o sonori e di ottenere questo messaggio di errore:

```
Chiamata di funzione non valida

<OK>   <Guida>
```

QBasic evidenzierà l'enunciato che non è stato in grado di eseguire. Per esempio, se si cerca di eseguire il programma precedente con modalità schermo 0, QBasic interrompe il programma, ritorna nella Finestra di digitazione e mostra questa finestra di dialogo per poi evidenziare l'enunciato PSET. Perché?

Se date un'occhiata alla tabella 11-2 noterete che la modalità schermo 0 è una modalità di solo testo, con cui è impossibile usare enunciati grafici.

Altre condizioni che producono un errore simile sono le seguenti:

- Usare un valore per l'enunciato COLOR non supportato dalla modalità schermo.
- Usare un valore per l'enunciato LOCATE uguale a 0 o maggiore nel numero di righe e colonne disponibili.
- Usare un valore per l'enunciato SCREEN che l'adattatore video non supporta.

Se ricevete questo messaggio di errore è consigliabile rivedere la tabella 11-2, esaminare l'elenco delle modalità valide ottenuto col programma TEST-VID.BAS e poi apportare le correzioni necessarie.

la. Il valore dell'argomento *primo piano* determina il colore del testo e delle forme grafiche; può essere cambiato in un qualsiasi punto del programma senza che il cambiamento si rifletta sul testo o le figure già visualizzate sullo

schermo. Il valore dell'argomento *sfondo* specifica il colore dello sfondo (questo può essere modificato solo nelle modalità 0, 7, 8, 9, 10).

Osservate che la modifica del colore dello sfondo non ha un effetto immediato; per convertire l'intero sfondo al nuovo colore bisogna includere nel programma un'istruzione per l'aggiornamento del colore dello sfondo. L'enunciato CLS è molto utile a questo scopo anche se è consigliabile usarlo all'inizio del programma per evitare che cancelli il testo o la grafica già visualizzata sullo schermo.

L'enunciato PRESET

L'enunciato PRESET funziona allo stesso modo di PSET, tranne che per un importante aspetto. Se l'argomento colore non viene specificato, QBasic mostra il pixel nel colore dello sfondo, il che permette di cancellare dei pixel semplicemente non specificando questo argomento. L'enunciato PRESET presenta la seguente sintassi:

`PRESET (`*`coordinataX`*`, `*`coordinataY`*`)[, `*`colore`*`]`

Notate che l'enunciato PRESET cancella qualunque cosa si trovi in corrispondenza a *(coordinataX, coordinataY)*. Per esempio, se si usa un enunciato PRINT per stampare un messaggio e poi si usa un enunciato PRESET le cui coordinate coprono parzialmente l'area in cui si trova il messaggio, è possibile che il messaggio venga tagliato.

Esercizio: Uso dell'enunciato PRESET

1. Scrivete il programma PRESET.BAS (figura 11-8).
2. Eseguite il programma. Se il vostro adattatore video lo permette specificate il valore 1, 7 o 13 per ottenere il massimo effetto. I commenti nel programma mostrano cosa bisogna cambiare per trarre il maggior vantaggio da modalità ad alta risoluzione.
   Durante l'esecuzione del programma potete osservare un interessante effetto di PRESET. Appena prima che cominci il "bombardamento" il programma stampa il messaggio "Premete un tasto qualsiasi per terminare", messaggio che viene poi demolito dal "bombardamento stesso".
   Notate l'uso della funzione INKEY$, che appare in questo programma per la prima volta. La funzione controlla la tastiera per verificare se qualche tasto è stato premuto; in questo caso INKEY$ restituisce il codice ASCII associato con il tasto, altrimenti restituisce una stringa vuota. A

```
' PRESET.BAS
' Questo programma mostra l'uso dell'enunciato PRESET

CLS

INPUT "Inserite una modalità schermo (0-13): ", modoNum%
INPUT "Premete Invio per iniziare il bombardamento...", richiesta$

SCREEN modoNum%
PRINT "Premete un tasto qualsiasi per terminare"

DO
   RANDOMIZE TIMER
   randCol% = INT(RND(1) * 320)    ' Numero di colonna casuale per
                                   ' il "bombardamento";
   ' suppone che vi sia una risoluzione orizzontale di 320;
   ' va modificato per modalità con risoluzioni più elevate

   FOR i% = 1 TO 199  ' suppone che vi sia una risoluzione
                      ' verticale di 200  va modificato per modalità
                      ' con risoluzioni più elevate
      PSET (randCol%, i%)
      PRESET (randCol%, i% - 1)
      FOR j% = 1 TO 35   ' ritarda il loop
      NEXT j%

   NEXT i%
LOOP UNTIL INKEY$ <> ""
```

Figura 11-8. *Una dimostrazione dell'enunciato PRESET.*

differenza di INPUT e LINE INPUT la funzione INKEY$ non aspetta che l'utente prema Invio per continuare.

Posizionamento dei pixel tramite coordinate

Si possono usare due sistemi di coordinate per posizionare i pixel sullo schermo: *coordinate assolute* e *coordinate relative*.

Coordinate assolute

Il sistema di coordinate usato fino ad ora è il sistema di coordinate assolute. In questo sistema tutti valori delle coordinate sono basati su quelle dell'angolo superiore sinistro, (0, 0). Per posizionare un pixel sullo schermo bisogna perciò calcolare (o indovinare) il numero di colonne a destra e il numero di righe in basso, individuando in questo modo i valori delle coordinate di quel pixel.

Questo sistema funziona molto bene ma se si vogliono usare singoli pixel per creare una forma particolare si va incontro a una grande quantità di calcoli. Il sistema di coordinate relative semplifica di molto il lavoro.

Coordinate relative

Con questo sistema le coordinate definite per uno specifico pixel sono relative all'ultimo pixel posto sullo schermo.

La figura che segue mostra i due diversi sistemi: per collocare il primo pixel si devono usare coordinate assolute perché non esiste alcuna posizione di riferimento; per il secondo e i successivi si possono usare le coordinate relative.

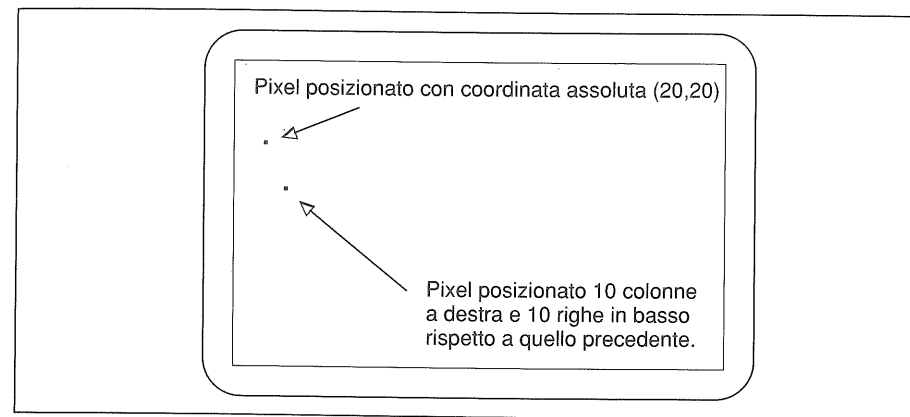
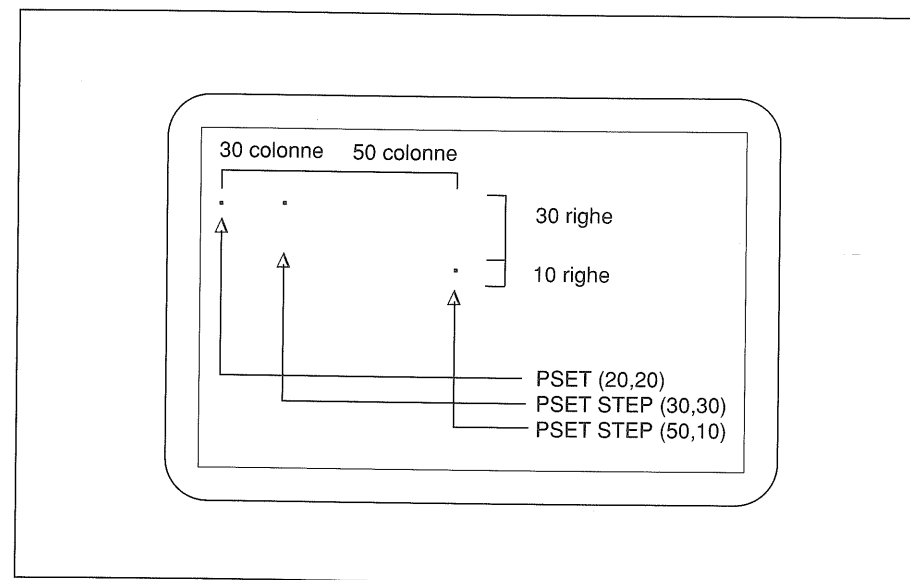
Per usare il sistema di coordinate relative ci si serve della parola chiave STEP usata con gli enunciati PSET e PRESET.

La parola chiave STEP

La sintassi di un enunciato PSET che contiene la parola chiave STEP è la seguente:

```
PSET STEP (coordinataX, coordinataY)[, colore]
```

Notate che la sintassi è identica a quella di un enunciato PRESET.

I valori *coordinataX* e *coordinataY* sono numeri positivi o negativi che dicono a QBasic dove posizionare il pixel in relazione a quello precedente posizionato con un enunciato PSET o PRESET. Se non esiste alcun pixel, QBasic usa le coordinate assolute (0, 0).

La figura che segue mostra questo effetto a catena: la posizione di ogni pixel, tranne il primo , dipende dalla posizione del pixel precedente. Ricordate sempre che la parola chiave STEP posiziona un pixel relativamente alla posizione dell'ultimo pixel posizionato.

Esercizio: Uso della parola chiave STEP

1. Scrivete il programma STEP-1.BAS (figura 11-9).


```
' STEP-1.BAS
' Questo programma mostra l'uso della parola chiave STEP dando
' l'illusione di muoversi nello spazio

CLS

CONST DELAY% + 200 ' controlla la velocità con cui le stelle appaiono
INPUT "Inserite una modalità schermo (0-13): ", modoNum%
SCREEN modoNum%

DO
   PSET (160, 100), 0 ' imposta al centro (risoluzione
   ' presunta di 320*200)
   quad% = INT(RND(1) * 4)  ' numero scelto a caso per il quadrante
   ' in cui la "stella" apparirà
   randX% = INT(RND(1) * 10)    ' numero scelto a caso per il movimento
                                ' relativo alla colonna
   randY% = INT(RND(1) * 10)    ' numero scelto a caso per il movimento
                                ' relativo alla riga

   IF quad% = 0 OR quad% = 1 THEN      ' il movimento normale è verso
                                       ' il basso
      randY% = -randY%   ' valore opposto di Y per movimento verso
                         ' l'alto
   END IF
   IF quad% = 0 OR quad% = 3 THEN ' Il movimento normale è a destra
      randX% = - randX%  ' Valore opposto di Y per movimento
                         ' verso sinistra
   END IF

   FOR i% = 1 TO 20
      PSET STEP(-randX% * i%, randY% * i%)  ' disegna una "stella"

      FOR j% = 1 TO DELAY%  ' ritarda il loop
      NEXT j%
      PRESET STEP(0, 0)  ' cancella la "stella"

   NEXT i%
LOOP UNTIL INKEY$ <> ""
```

Figura 11-9. *Viaggiare nello spazio con la parola chiave STEP.*

2. Eseguite il programma e specificate un valore per lo schermo. Poiché il programma presuppone una risoluzione di 320*200 cercate, se possibile, di usare un valore pari a 1, 7 o 13. Il programma può essere interrotto premendo un tasto qualsiasi.

Questo programma dà l'illusione di viaggiare nello spazio simulando il movimento delle "stelle". All'interno del loop DO il programma imposta un pixel "invisibile" al centro dello schermo che rappresenta il punto di riferimento sul quale i successivi enunciati PSET basano le loro coordinate. Il numero casuale *quad%* determina in quale quadrante la stella apparirà. Notate i due enunciati IF (uno per i quadranti 0 e 1 e l'altro per i quadranti 0 e 3; il quadrante 2 usa valori positivi per cui nessuna modifica è necessaria): i valori modificati in questi enunciati si riflettono sulle coordinate di PSET nel loop FOR (alcune usano coordinate positive, altre negative, altre entrambe).

Nell'enunciato PSET un numero scelto a caso viene moltiplicato per il valore corrente della variabile loop *i%*. Dopo ogni loop il valore risultante aumenta, facendo in modo che la stella si allontani dal centro dello schermo, acquisendo velocità quando si avvicina ai limiti dello schermo stesso.

Infine, notate come viene usato l'enunciato PRESET: la parola chiave STEP fa in modo che PRESET usi coordinate relative (coordinate determinate dall'enunciato PSET). Poiché le coordinate fornite sono (0, 0), l'enunciato PRESET forza QBasic a collocare i pixel PRESET nella stessa posizione dei pixel PSET che li precedono, cancellandoli.

CREAZIONE DI FORME COMPLESSE

La possibilità di definire singoli pixel fornisce un controllo molto elevato sulla creazione di grafici nello schermo di output. Tuttavia, per forme complesse quali righe, riquadri, cerchi e poligoni le coordinate da calcolare possono essere moltissime; nel caso di una riga si potrebbe usare un loop per definire i punti individuali ma, nonostante questo, sarebbe necessaria una gran quantità di calcoli. Per nostra fortuna QBasic fornisce una serie di strumenti che facilitano la creazione di forme complesse.

L'enunciato LINE

L'enunciato LINE serve alla creazione di linee senza richiedere un grande sforzo da parte vostra.

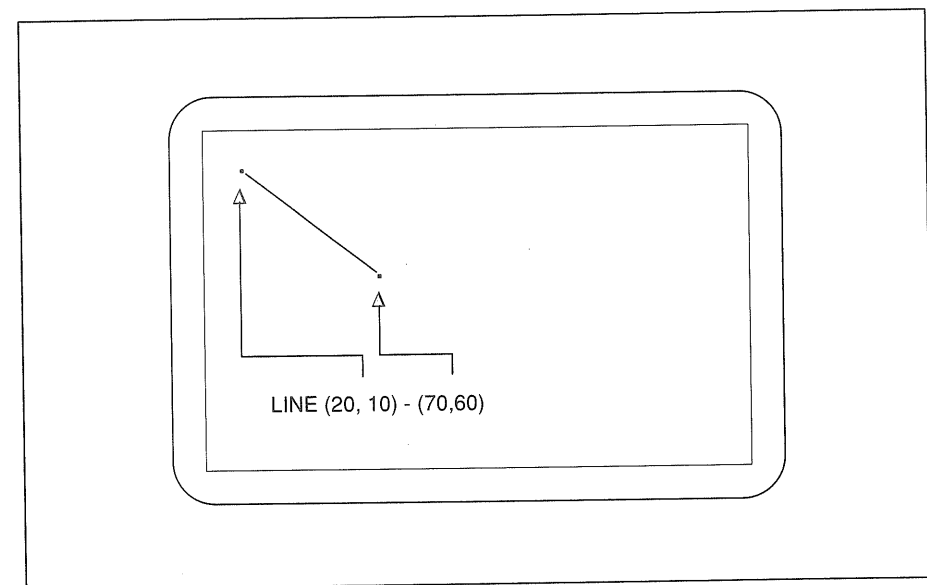
Disegnare una linea semplice

Nella sua forma più semplice l'enunciato LINE presenta questa sintassi:

```
LINE (x1, y1) - (x2, y2) [, colore]
```

Notate che, a differenza di PSET e PRESET, che usano una sola coppia di coordinate, l'enunciato LINE ne richiede due:

- *x1* e *y1* sono le coordinate di riga e colonna del punto di inizio della linea.
- x2 e y2 sono le coordinate di riga e colonna del punto in cui termina la linea.

Questo esempio mostra come funziona l'enunciato LINE:

Si noti che il punto di inizio non deve necessariamente trovarsi alla sinistra del punto di fine. È importante specificare le coordinate lasciando poi che QBasic riempia i punti compresi tra i due estremi.

Come per altri comandi grafici, l'intervallo di valori validi per *x1*, *y1*, *x2*, e *y2* dipende dalla modalità dell'adattatore video. Le coordinate vanno racchiuse tra parentesi e separate da un trattino.

Esercizio: Uso dell'enunciato LINE

1. Scrivete il programma LINE-1.BAS (figura 11-10).

```
' LINE-1.BAS
' Questo programma mostra l'uso dell'enunciato LINE

CLS

INPUT "Inserite una modalità schermo (0-13): ", modoNum%
INPUT "Quanti colori? ", numColori%
SCREEN modoNum%

CONST DELAY% = 100     ' controlla il ritardo dopo il
      ' disegno di ogni riga

DO
      tono% = INT(RND(1) * numColori%) + 1     ' colore della linea
      x1pos% = INT(RND(1) * 320)   '  coordinate di inizio
      y1pos% = INT(RND(1) * 200)
      x2pos% = INT(RND(1) * 320)   '  coordinate di fine
      y2pos% = INT(RND(1) * 200)
      LINE 9x1pos%, y1pos%) - (x2pos%, y2pos%), tono%

      FOR i% = 1 TO DELAY%   " ritarda il loop
      NEXT i%
LOOP WHILE INKEY$ = ""
```

Figura 11-10. *Un programma che disegna righe a caso attraverso l'enunciato LINE.*

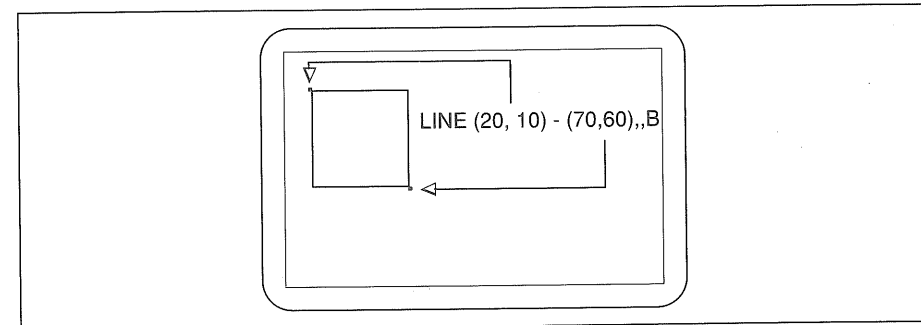
2. Il programma chiede di definire una modalità schermo e il massimo numero di colori per poi stampare delle linee sullo schermo, a caso e in colori diversi. Premete un tasto qualsiasi per uscire dal programma.

Disegnare un riquadro

Anche disegnare un riquadro, sia vuoto o pieno, è molto semplice. Si potrebbero usare quattro enunciati LINE, ma QBasic dispone di una particolare funzionalità che permette di risparmiare tempo e ridurre i calcoli necessari.

Per creare un riquadro si usa ancora l'enunciato LINE con una sintassi modificata:

`LINE (x1, y1) - (x2, y2)[, [colore][, [B[F]]]]`

La differenza rispetto alla sintassi vista in precedenza sta nell'uso delle opzioni B e F. Se si usa B, QBasic disegna una linea usando le coordinate iniziali e finali come angoli opposti del riquadro, in questo modo:

Notate che QBasic non disegna una riga tra i punti di inizio e fine come farebbe nella caso di un enunciato LINE semplice.

Se invece si usa l'opzione F, subito dopo B, QBasic riempie il riquadro con il colore scelto per il primo piano. Questa opzione può essere usata solo insieme a B.

Esercizio: Riquadri pieni e vuoti

1. Scrivete il programma LINE-2.BAS (figura 11-11).

2. Al momento dell'esecuzione il programma crea una serie casuale di riquadri vuoti o pieni, in base alle vostre risposte. I riquadri possono essere di qualsiasi dimensione o forma e possono essere collocati dovunque sullo schermo.

Disegnare forme complesse

Le linee e i riquadri pieni e vuoti visti fino ad ora erano basati sull'uso di coordinate assolute. Tuttavia, anche con l'enunciato LINE (come per PSET e PRESET) è possibile l'uso della parola chiave STEP per specificare coordinate relative anche nel disegno di un riquadro. Si può anche omettere il punto di inizio della linea o del riquadro, costringendo QBasic a usare le coordinate

```
' LINE-2.BAS
' Questo programma mostra la funzionalità dell'enunciato
' LINE nel disegno di riquadri.

CLS

INPUT "Inserite una modalità schermo (0-13): ", modoNum%
INPUT "Quanti colori? ", numColori%
INPUT "Riquadri pieni o vuoti (P o V)? ", quadro$
SCREEN modoNum%

CONST DELAY% = 800

DO
   tono% = INT(RND(1) * numColori%) + 1  ' colore della linea
   x1pos% = INT(RND(1) * 320)   ' coordinate di partenza
   y1pos% = INT(RND(1) * 200)
   x2pos% = INT(RND(1) * 320)   ' coordinate di arrivo
   y2pos% = INT(RND(1) * 200)
   IF UCASE$(quadro$) = "P" THEN
      LINE (x1pos%, y1pos%,) - (x2pos%, y2pos%), tono%, B
   ELSEIF UCASE$(quadro$) = "V" THEN
      LINE (x1pos%, y1pos%) - (x2pos%, y2pos%), tono%, BF
   END IF

   FOR i% = 1 TO DELAY%  ' Ritarda il loop
   NEXT i%

LOOP UNTIL INKEY$ <> ""
```

Figura 11-11. *Disegno di riquadri a caso con l'enunciato LINE.*

del punto finale dell'ultimo oggetto disegnato come punto iniziale per la nuova linea.

La sintassi dell'enunciato LINE con l'uso di coordinate relative è la seguente:

LINE [[STEP](*x1*, *y1*)] - [STEP](*x2*, *y2*)[, [*colore*][, [B[F]]]]

L'uso di coordinate relative permette di creare disegni complessi risparmiando tempo. La figura 11-12 mostra alcuni esempi di enunciati LINE che usano coordinate assolute e relative e i loro risultati. Notate che si presuppone che il punto finale sia impostato a (20, 20).

Enunciato LINE	Risultato
LINE — (30, 50)	Disegna da (20, 20) a (30, 50)
LINE — STEP(30, 50)	Disegna da (20, 20) a (50, 70)
LINE(40, 40) — STEP(60, 60)	Disegna da (40, 40) a (100, 100)
LINE STEP(30, 50) — (70, 70)	Disegna da (50, 70) a (70, 70)
LINE STEP(30, 50) — STEP(70, 70)	Disegna da (50, 70) a (120, 140)

Figura 11-12. *Risultati di enunciati LINE usate con coordinate assolute e relative.*

Dalla figura 11-12 si possono trarre alcune conclusioni:

- Se si omettono le coordinate di inizio QBasic usa, come coordinate di inizio, quelle del punto più recente.
- Se si omettono le coordinate di inizio e si usa la parola chiave STEP con le coordinate di fine, QBasic posiziona il punto finale *in relazione* alle coordinate del punto più recente.
- Se si includono le coordinate di inizio e si usa la parola chiave STEP con le coordinate di fine, QBasic posiziona il punto finale *in relazione* alle coordinate di inizio specificate.
- Se si usa la parola chiave STEP con le coordinate di inizio e non la si usa con quelle di fine, QBasic posiziona le coordinate di inizio *in relazione* alle coordinate del punto più recente e posiziona il punto di fine alle coordinate *assolute* specificate.
- Se si usa la parola chiave STEP con le coordinate di inizio e di fine, QBasic posiziona le coordinate di inizio *in relazione* alle coordinate del punto più recente e posiziona il punto finale *in relazione* alle coordinate del punto iniziale appena calcolato.

L'uso di coordinate relative permette di collegare linee e riquadri, semplificando il disegno di forme complesse.

Esercizio: Lavorare con LINE e coordinate relative

1. Scrivete il programma STEP-2.BAS (figura 11-13).
2. Eseguite questo semplice programma che inizia disegnando un pixel alle coordinate (150, 50), punto d'inizio del disegno. Buona parte del lavoro viene fatta dal loop DO. Il programma chiede di definire i valori orizzon-

```
' STEP-2.BAS
' Questo programma mostra l'uso della parola chiave
' STEP in un enunciato LINE

CLS

INPUT "Inserite una modalità schermo (0-13): ", modoNum%
SCREEN modoNum%

vuoto$ = SPACE$(39)

PRINT "Digitare 0 e 0 per uscire"
PSET (150, 50)    ' stabilisce un punto d'inizio

DO
      LOCATE 22, 1
      INPUT "Spostamento orizzontale (+ o -): ", oriz%
      INPUT "Spostamento verticale (+ o -): ", vert%
      LINE -STEP(oriz%, vert%)
      LOCATE 22, 1
      PRINT vuoto$
      PRINT vuoto$
LOOP UNTIL oriz% = 0 AND vert% = 0
```

Figura 11-13. *Un semplice programma di disegno.*

tali e verticali, che possono essere sia positivi che negativi. Dopo aver acquisito questi valori il programma usa l'enunciato LINE con STEP per disegnare la linea.

L'enunciato CIRCLE

QBasic permette anche di creare dei cerchi. Con l'enunciato CIRCLE si possono creare cerchi di varie dimensioni che, se usati con altri strumenti grafici, possono dar vita a figure utili e interessanti.

Nella sua forma più semplice l'enunciato CIRCLE presenta questa sintassi:

CIRCLE (*coordinataX*, *coordinataY*), *raggio*[, *colore*]

coordinataX e *coordinataY* sono le coordinate di schermo orizzontale e verticale già viste. Questi valori dicono a QBasic dove posizionare il centro del cerchio.

raggio è il valore che specifica, in pixel, il raggio del cerchio (il raggio di un cerchio è la metà del diametro). *colore* è un intero che dice a QBasic di che colore dev'essere il cerchio; se questo valore non viene specificato QBasic usa il colore di primo piano corrente.

Come per gli altri enunciati grafici le coordinate usate dipendono interamente dalla modalità grafica impostata con l'enunciato SCREEN.

Esercizio: Uso dell'enunciato CIRCLE

1. Scrivete il programma CIRCLE-1.BAS (figura 11-14).
2. Eseguitelo, specificando la modalità schermo. Il programma chiede di specificare un valore di colonna, uno di riga e uno per il raggio; l'enunciato CIRCLE disegna un cerchio in base ai valori specificati. Poi, quando si preme Invio per continuare, l'enunciato CLS ripulisce lo schermo e il processo si ripete fino a quando si digita f o F.
   Provate a inserire dei valori più grandi delle coordinate massime dello schermo o a specificare un raggio molto ampio. Questa tecnica può dar vita a effetti molto particolari come la creazione di archi su un lato dello schermo.

```
' CIRCLE-1.BAS
' Questo programma mostra l'uso dell'enunciato CIRCLE
```

Disegnare archi

L'enunciato CIRCLE permette anche di disegnare archi (segmenti di un cerchio). In questo caso l'enunciato CIRCLE va usato con i valori di inizio e di fine:

CIRCLE (*coordinataX*, *coordinataY*), *raggio*[, [*colore*][, [*inizio*][*fine*]]]

La sintassi è simile a quella precedente ma con l'aggiunta dei valori di inizio e fine. I valori di *inizio* e *fine* sono le misure, in radianti, del punto iniziale e finale dell'arco.

Sebbene sia più comune misurare gli archi in gradi l'enunciato CIRCLE richiede che le misure di inizio e fine vengano fornite in radianti. La figura 11-15 mostra la relazione tra il sistema di misurazione in gradi e quello in ra-

```
CLS

INPUT "Inserite una modalità schermo (0-13): ", modoNum%
SCREEN modoNum%

DO
        LOCATE 21, 1
        INPUT "Inserite il numero di colonna: ", colNum%
        INPUT "Inserite il numero di riga: ", rigaNum%
        INPUT "Inserite il raggio: ", raggio%
        CIRCLE (colNum%, rigaNum%), raggio%
        INPUT "Premete Invio per continuare o F per finire: " richiesta$
        CLS
LOOP UNTIL UCASE$(richiesta$) = "F"
```

Figura 11-14. *Un semplice programma per il disegno di cerchi.*

dianti. Potete usare questa figura come riferimento per determinare il punto di inizio e fine dell'arco da disegnare (infatti sono state incluse informazioni sulla conversione, che possono essere molto utili).

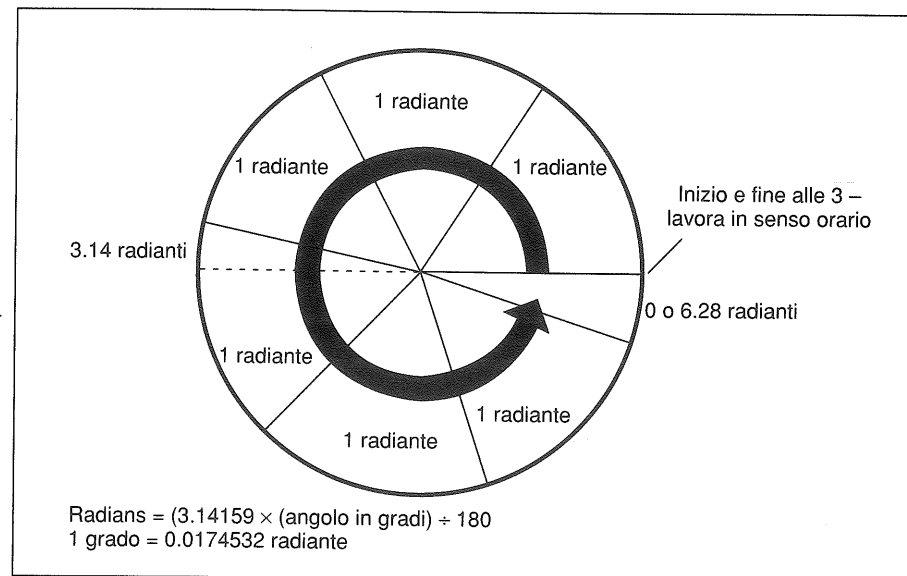



Figura 11-15. *Misurazione di un arco in radianti.*

Le misurazioni iniziano da 0 (ore 3) e progrediscono in senso orario fino a un massimo di 6.28 radianti; il punto iniziale (0) e finale (6.28) coincidono.

I valori iniziali e finali indicati sono numeri compresi tra 0 e 6.28 (questo valore è tecnicamente 2 * pi greco; per semplicità usiamo dei numeri arrotondati).

Se si mette un segno meno davanti al valore iniziale o finale, QBasic disegna una riga (il raggio) da quel punto sull'arco fino al centro del "cerchio".

Esercizio: Lavorare con gli archi

1. Scrivete il programma CIRCLE-2.BAS (figura 11-16).

```
' CIRCLE-2.BAS
' Questo programma  mostra come creare degli archi con l'enunciato CIRCLE

CLS

INPUT "Inserite una modalità schermo (0-13): ", modoNum%

SCREEN modoNum%

DO
   CLS
   PSET (150, 30): PSET (150, 125)      ' disegna i punti superiore
                                        ' e inferiore
   PSET (95, 78): PSET(205, 78)         ' disegna i punti sinistro/destro
   LOCATE 2, 18: PRINT "1.57"           ' etichetta del punto superiore
   LOCATE 10, 7: PRINT "3.14"           ' etichetta del punto sinistro
   LOCATE 18, 18: PRINT "4.71"          ' etichetta del punto inferiore
   LOCATE 10, 28: PRINT "0 o 6.28"      ' etichetta del punto destro

   LOCATE 20, 1
   PRINT "Inserite dei valori tra 0 e 6.28"
   PRINT "(Un valore negativo disegna un raggio)"
   INPUT "Inserite un punto di inizio: ", inizio!
   INPUT "Inserite un punto di fine: ", fine!
   CIRCLE (150, 78), 50, , inizio!, fine!  ' nessun valore per il colore
   INPUT "Premete Invio per continuare o F per finire: " richiesta$

LOOP UNTIL UCASE$(richiesta$) = "F"
```

Figura 11-16. *Un programma per la creazione di archi.*

2. Eseguite il programma e sperimentate la creazione di archi, inserendo i valori di inizio e fine (da 0 a 6.28) e mettendo un segno meno davanti a uno dei numeri se volte che QBasic disegni una linea da quel punto al centro del cerchio). Notate che il valore di inizio può essere più piccolo del valore di fine. Provate varie combinazioni di valori per vedere i diversi effetti.

UN ULTIMO SGUARDO ALLA GRAFICA IN QBASIC

Abbiamo visto che QBasic offre un gruppo di strumenti di disegno utili e potenti. Purtroppo non siamo in grado di analizzarli tutti in questo libro ma quelli visti sono più che sufficienti per cominciare a imparare la programmazione grafica e per inserire effetti particolari nei vostri programmi:

- Grafici a barre che rappresentano graficamente dei dati.
- Grafici a torta che mostrano la distribuzione dei dati in diverse categorie.
- Figure che rendono più interessanti i programmi.
- Programmi per proteggere lo schermo.
- Animazioni per giochi e simulazioni.

NOTA: La guida in linea di QBasic è in grado di fornire ulteriori informazioni sulla programmazione grafica.

INTRODUZIONE ALLA PROGRAMMAZIONE SONORA

Abbiamo visto come usare grafici per ravvivare i programmi; un altro modo per rendere i vostri programmi più "vivaci" è quello di sfruttare le funzionalità sonore del vostro computer.

Ancora sull'enunciato SOUND

Nel capitolo 6 abbiamo brevemente introdotto l'enunciato SOUND, parlando di loop. Cercheremo ora di approfondire l'argomento.

L'enunciato SOUND presenta questa sintassi:

SOUND *frequenza, durata*

frequenza è un valore intero da 37 a 32767 che indica la frequenza, in hertz (cicli al secondo) del tono desiderato; basse frequenze creano dei toni bassi e viceversa. Notate che l'orecchio umano è in grado di percepire le frequenze tra i 20 e i 20000 hertz e che i toni con frequenze sopra i 16000 hertz sono più facilmente udibili dai cani che dalla maggior parte delle persone. La figura 11-17 elenca i valori di frequenza standard e le note loro associate dall'American Standards Association nel 1936.

durata è un valore intero che indica a QBasic la durata di una certa nota; questo valore specifica la durata in termini di "segmenti di secondo" (ci sono 18.2 segmenti in un secondo; perciò un durata di 9.1 indicherebbe a QBasic di suonare una certa nota per mezzo secondo).

In un enunciato SOUND è necessario specificare sia la frequenza che la durata.

Nei programmi visti fino ad ora che comprendevano effetti sonori non c'era alcuna intenzione di creare delle melodie ma è possibile abbinare i valori di frequenza alle note musicali scelte. Siccome il più piccolo valore di frequenza utilizzabile con SOUND è 37, la nota più bassa che potete suonare è RE#1.

Il numero a fianco di ciascuna nota ne indica l'ottava. Potete notare che per alzare una nota di un'ottava è sufficiente raddoppiarne la frequenza (approssimativamente). Per esempio DO2 ha una frequenza di 65.41 hertz e DO3 una frequenza di 130.81, che è circa il doppio di DO2. Ciò può essere utile quando si vuole aggiungere della musica ai programmi, perché basta raddoppiare la frequenza per aumentare una nota di un'ottava.

Esercizio: Uso dell'enunciato SOUND

1. Scrivete il programma SOUND-1.BAS (figura 11-18).
2. Eseguite il programma e inserite un valore di ottava. Notate che la tabella delle frequenze nell'enunciato DATA comincia con DO2, per cui, se inserite un valore di ottava pari a 1, il programma esegue una scala maggiore da DO2 a DO3. Siccome l'enunciato SOUND accetta solo valori interi come frequenze, l'enunciato DATA contiene valori arrotondati all'intero più vicino.

Nota	Frequenza	Nota	Frequenza	Nota	Frequenza
RE#1	38.89	SOL3	196.00	LA#5	932.33
MI1	41.20	SOL#3	207.65	SI5	987.77
FA1	43.65	LA3	220.00	DO6	1046.50
FA#1	46.25	LA#3	233.08	DO#6	1108.73
SOL1	49.00	SI3	246.94	RE6	1174.66
SOL#1	51.91	DO4	261.63	RE#6	1244.51
LA1	55.00	DO#4	277.18	MI6	1328.51
LA#1	58.27	RE4	293.66	FA6	1396.91
SI1	61.74	RE#4	311.13	FA#6	1479.98
DO2	65.41	MI4	329.63	SOL6	1567.98
DO#2	69.30	FA4	349.23	SOL#6	1661.22
RE2	73.42	FA#4	369.99	LA6	1760.00
RE#2	77.78	SOL4	392.00	LA#6	1864.66
MI2	82.41	SOL#4	415.30	SI6	1975.53
FA2	87.31	LA4	440.00	DO7	2093.00
FA#2	92.50	LA#4	466.16	DO#7	2217.46
SOL2	98.00	SI4	493.88	RE7	2349.32
SOL#2	103.83	DO5	523.25	RE#7	2489.02
LA2	110.00	DO#5	554.37	MI7	2637.02
LA#2	116.54	RE5	587.33	FA7	2793.83
SI2	123.47	RE#5	622.25	FA#7	2959.96
DO3	130.81	MI5	659.26	SOL7	3135.96
DO#3	138.59	FA5	698.46	SOL#7	3322.44
RE3	146.83	FA#5	739.99	LA7	3520.00
RE#3	155.56	SOL5	783.99	LA#7	3729.31
MI3	164.81	SOL#5	830.61	SI7	3951.07
FA3	174.61	LA5	880.00	DO8	4186.01
FA#3	185.00				

Figura 11-17. *Frequenze delle note musicali comprese nell'estensione di circa sette ottave.*

Notazione musicale in QBasic

La maggior parte di noi non ha grande familiarità con le note musicali in termini di frequenza, ma QBasic richiede che gli venga indicato quale nota suonare attraverso la frequenza assegnata a quella nota.

Se pensate di aggiungere degli effetti musicali ai vostri programmi avrete la necessità di convertire le note musicali nelle frequenze loro associate. A questo fine può essere utile creare una tabella di conversione.


```
' SOUND-1.BAS
' Questo programma mostra l'uso dell'enunciato SOUND

CLS

INPUT "Inserite il valore di un'ottava (1-7): ", ottava%
' suona un'ottava

FOR i% = 1 TO 8
   READ nota%   ' legge la frequenza di una nota
   nota% = nota% * (2 ^ ottava%)   ' aumenta la nota all'ottava
                                   ' desiderata
   SOUND nota%, 12 ' suona la nota
NEXT i%

' frequenze delle note piú importanti nella seconda ottava
DATA 65, 73, 82, 87, 98, 110, 123, 131
```

Figura 11-18. *Un programma che esegue una scala di note.*

Lavorare da spartito

Sebbene non rientri nello scopo di questo libro spiegare la notazione musicale in dettaglio, i suggerimenti che seguono possono essere utili quando si vuole aggiungere effetti musicali ai programmi. Diamo un'occhiata alla figura 11-19 che mostra i nomi e le posizioni delle note.

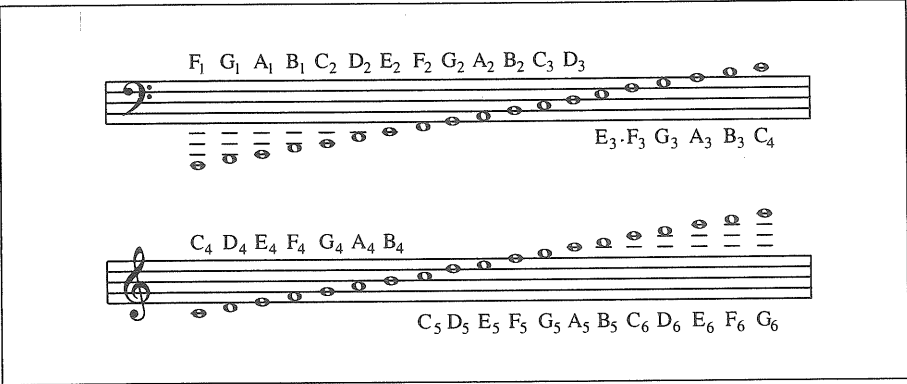
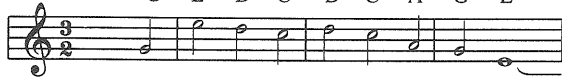
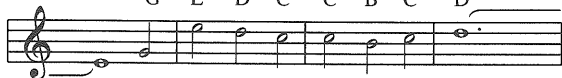



Figura 11-19. *Nomi e posizioni delle note sul pentagramma.*

Se lavorate partendo da uno spartito potete abbinare le vostre note con quelle della figura e poi verificare la frequenza associata nella figura 11-17. Vi suggeriamo di trascrivere nel vostro foglio musicale la frequenza associata a ogni nota.

Per individuare i valori di durata da inserire nell'enunciato SOUND basta decidere quanto a lungo si vuole che duri una nota di un intero e poi dividere il valore per 2 per avere la metà, per 4 per avere il quarto e così via. Dopo aver stabilito questi valori potete riportarli nel vostro spartito accanto a ciascuna nota.

Dopo aver determinato i valori di frequenza e durata per tutte le note potrete aggiungere la melodia al vostro programma.

Esercizio: Suonare una canzone con QBasic

La figura 11-20 mostra la musica della canzone americana *My Bonnie Lies Over the Ocean.* Dopo aver calcolato frequenza e durata abbiamo scritto un breve programma per eseguire questa canzone.

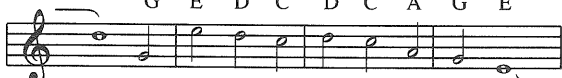
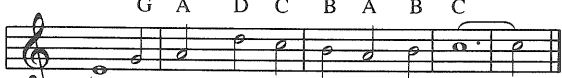



Figura 11-20. *Musica per My Bonnie Lies Over the Ocean.*

1. Scrivete il programma SOUND-2.BAS (figura 11-21).

```
' SOUND-2.BAS
' Questo programma mostra come suonare una canzone in QBasic

CLS

INPUT "Premete Invio per cominciare...", richiesta$

FOR i% = 1 TO 34
   READ nota%, durata%
   SOUND nota%, durata%
NEXT i%

'  My       bon-     nie      lies     o-       ver      the
DATA 392, 8, 659, 8,  587, 8,  523, 8,  587, 8,  523, 8,  440,8

'  o-       cean,    My       bon-     nie      lies
DATA 392, 8, 330, 32, 392, 8,  659, 8,  587, 8,  523, 8

'  o-       ver      the      sea;     My       bon-     nie
DATA 523, 8, 494, 8,  523, 8,  587, 40, 392, 8,  659, 8,  587, 8

'  lies     o-       ver      the      o-       cean--   O
DATA 523, 8, 587, 8,  523, 8,  440, 8,  392, 8,  330, 32, 392, 8

'  bring    back     my       bon-     nie      to       me!
DATA 440, 8, 587, 8,  523, 8,  494, 8,  440, 8,  494, 8,  523, 32
```

Figura 11-21. *Un programma che suona una canzone.*

2. Eseguite il programma e sentirete QBasic suonare la canzone. Si noti che i valori di frequenza e durata si alternano negli enunciati DATA. Nel loop FOR l'enunciato READ legge i due valori (uno per la frequenza e uno per la durata) sempre attraverso il loop.
   Esercitatevi con l'enunciato SOUND per vedere che controllo avete sui valori dell'enunciato DATA. Provate a moltiplicare *nota%* per 2 per alzare le note di un'ottava e a dividere *durata%* per 2 per osservare gli effetti sul tempo di esecuzione.

SOMMARIO

QBasic dispone di molti strumenti per la creazione di grafici e suoni (noi abbiamo visto solo i fondamentali). Dopo aver completato gli esercizi di questo libro sperimentate ulteriormente coi programmi visti in questo capitolo, usando la guida in linea per aver maggiori informazioni su questi argomenti. Grafici e suoni non sono argomenti di programmazione facili da apprendere ma sono in grado di rendere i vostri programmi molto più interessanti.

DOMANDE ED ESERCIZI

1. Quali sono i due componenti dell'apparecchiatura video del vostro computer e qual è la loro funzione?
2. Qual è la differenza tra modalità testo e grafica?
3. Qual è la funzione dell'enunciato LOCATE?
4. Scrivete un breve programma che chiede all'utente di inserire un nome per poi visualizzarlo sul lato sinistro dello schermo e "spostarlo" su quello destro.
5. Qual è la funzione dell'enunciato SCREEN?
6. Qual è la funzione degli enunciati PSET e PRESET e in cosa differiscono?
7. Qual è la differenza tra coordinate relative e assolute?
8. Vero o falso: per disegnare un riquadro pieno si può usare un enunciato LINE.
9. Vero o falso: per disegnare un cerchio pieno si può usare un enunciato CIRCLE .
10. Scrivete un programma che disegna un cerchio vicino al lato sinistro dello schermo e poi lo "sposta" a destra. (Un suggerimento: non dimenticate di cancellare il primo cerchio.)

CAPITOLO 12

Debugging dei programmi di QBasic

Fino ad ora abbiamo studiato molti aspetti importanti della programmazione in QBasic che costituiscono le basi per diventare dei programmatori esperti. Tuttavia succede spesso che un programma, anche se è stato scritto da programmatori esperti, si rifiuti di girare. In queste situazioni diventa importante diagnosticare e risolvere eventuali problemi ed errori di programmazione ed è qui che entra in gioco il concetto di debug.

Con *Debug* si intende il processo di individuazione e correzione di errori in un programma. L'abilità di individuare e correggere un problema in un programma è uno degli aspetti più importanti che un programmatore alle prime armi deve sviluppare. Il debug richiede la capacità di pensare in modo modulare (di analizzare un programma come farebbe un computer, controllandone l'esecuzione passo dopo passo) al fine di trovare il problema che impedisce al programma di svolgere il proprio compito.

Fortunatamente QBasic dispone di molti potenti comandi di debug che aiutano a esaminare i programmi. Questo capitolo introduce questi comandi e descrive come programmare in modo da ridurre al minimo gli errori. In particolare verrà mostrato il debug di un programma dall'inizio alla fine per sperimentare i comandi descritti.

IL PROCESSO DI DEBUG

Il debug di un programma consiste nel ripetere queste azioni fino a quando il programma viene eseguito con successo:

1. Eseguire il programma.
2. Osservare errori e punti problematici.
3. Isolare e studiare gli enunciati che hanno prodotto gli errori, usando gli strumenti di programmazione.
4. Correggere gli errori.

Se disponete di una stampante può essere utile stampare una copia di ogni programma di cui volete fare il debug; lo studio del codice è infatti uno degli aspetti principali della ricerca degli errori e la stampa su carta risulta spesso più facile da leggere.

Tre tipi di errori

Vi sono tradizionalmente tre tipi di errori: *errori di sintassi*, *errori run-time* ed *errori logici*.

- Un errore di sintassi è un errore di programmazione che viola le regole di QBasic (come un separatore mancante o una parola chiave scritta male). QBasic evidenzia gli errori di sintassi già in fase di digitazione e non permette l'esecuzione del programma fino a quando non sono stati tutti corretti.
- Un errore run-time è un errore che interrompe inaspettatamente il programma durante l'esecuzione. Gli errori run-time si verificano quando un evento esterno o un errore di sintassi non scoperto costringe il programma a fermarsi durante l'esecuzione; eccedere i limiti di una matrice o tentare di aprire un file con un nome errato sono esempi di errori run-time.
- Un errore logico è un errore umano (un errore di programmazione che forza il programma a dare dei risultati sbagliati). I maggior sforzi nel processo di debug sono concentrati nell'individuare errori logici introdotti dal programmatore.

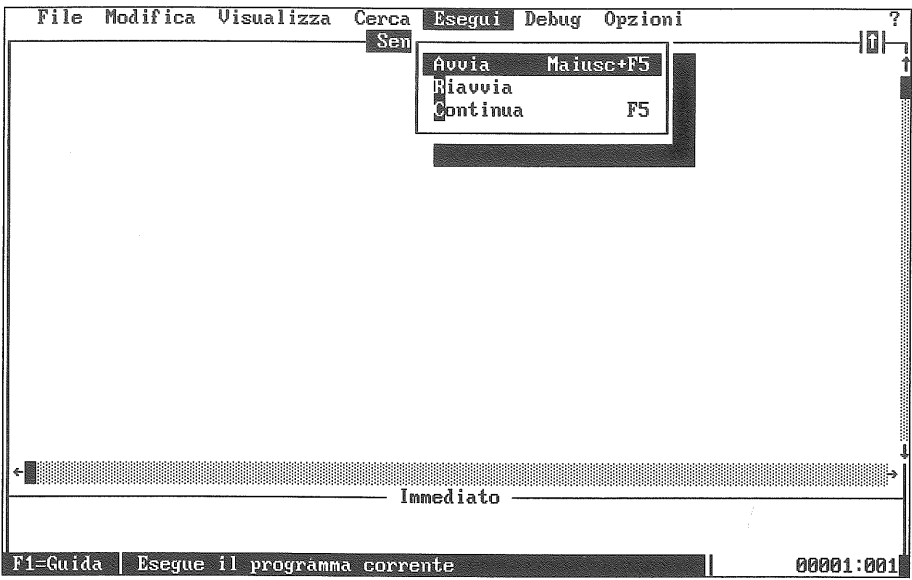
Ricordate di fare uso della Guida "on-line" quando incontrate dei messaggi di errore prodotti da errori di sintassi o run-time.

COMANDI DI MENU DI QBASIC

QBasic dispone di comandi di menu che aiutano a individuare i bug del programma. Nelle sezioni che seguono vedremo i comandi collegati al debugging all'interno dei menu Esegui, Debug, Visualizza e Opzioni. Per maggiori informazioni su questi comandi può essere utile consultare la Guida on-line.

Il menu Esegui

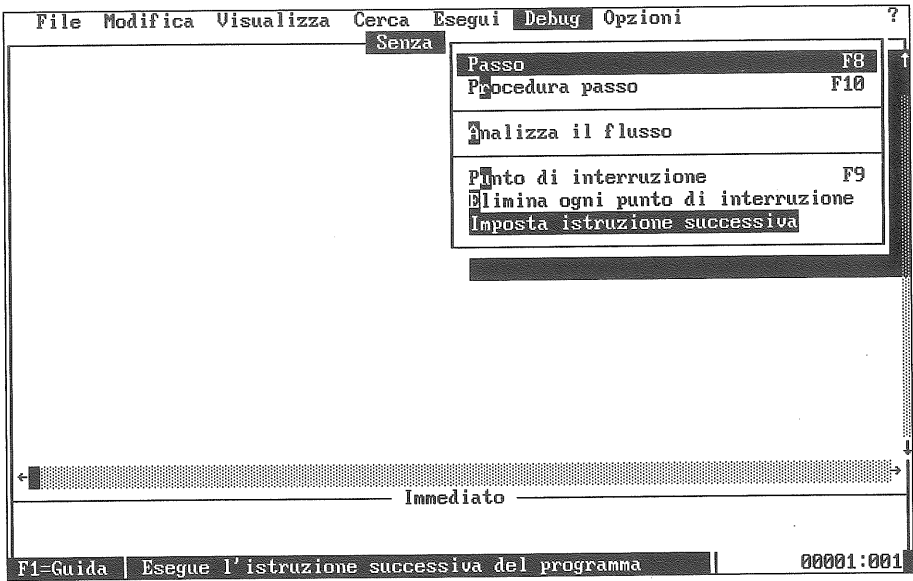
Il menu Esegui contiene tre comandi correlati al debug:

- Start (Maiusc-F5) esegue il programma attualmente in memoria dall'inizio e a velocità piena. Premete Ctrl-Break per interrompere il programma in fase di esecuzione.

- Riavvia reimposta tutte le variabili del programma e evidenzia il primo enunciato eseguibile. Usate questo comando per spostarvi all'inizio del programma prima di eseguire gli enunciati una alla volta coi comandi Passo e Procedura passo del menu Debug.
- Continua (F5) esegue il programma attualmente in memoria dalla riga corrente. Usate questo comando per continuare l'esecuzione del programma a velocità piena.

Il menu Debug

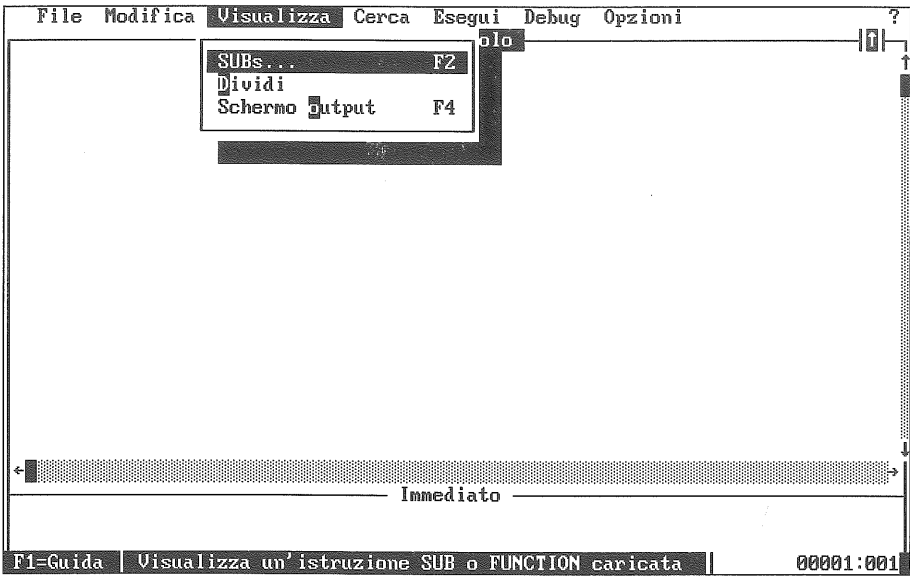
Il menu Debug contiene sei comandi correlati al debug:

- Passo (F8) esegue un enunciato ed evidenzia quello successivo eseguibile. Usate questo comando per eseguire il programma un'istruzione alla volta.
- Procedura Passo (F10) esegue un enunciato ed evidenzia quello successivo eseguibile. A differenza di Passo, questo comando esegue un intero sottoprogramma o una funzione definita dall'utente come un passo.
- Analizza il flusso è un comando interruttore che prepara il programma per un'esecuzione rallentata; quando è attivo, un programma viene eseguito lentamente e ogni suo enunciato viene evidenziato al momento del-

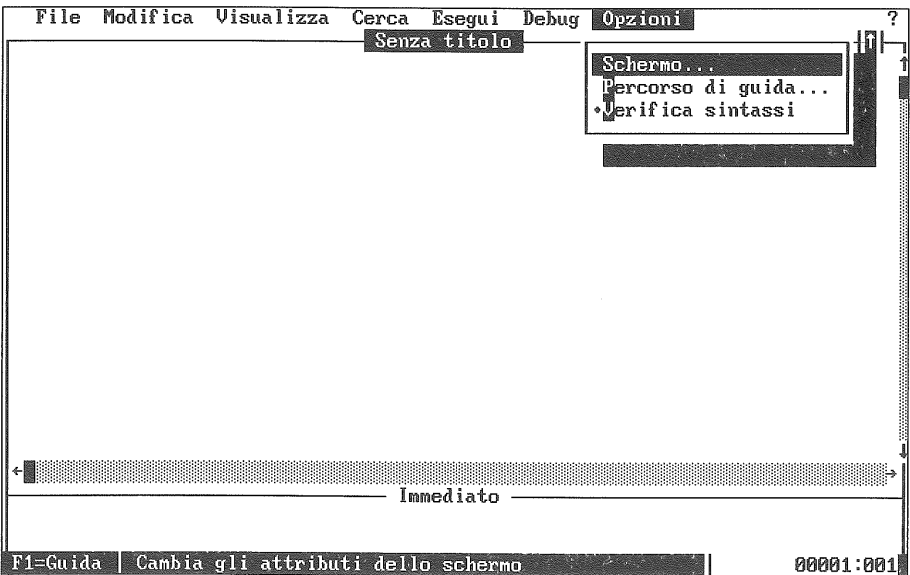
l'esecuzione; quando è disattivato il programma viene eseguito a velocità normale.

- Punto di interruzione (F9) imposta o cancella un breakpoint nel programma (una linea in corrispondenza alla quale il programma si interromperà) se disponete di un monitor a colori noterete che queste linee appaiono in rosso per definizione.
- Elimina ogni punto di interruzione cancella tutti i breakpoint impostati col comando precedente. Usate questo comando per cancellare più breakpoint allo stesso tempo.
- Imposta istruzione successiva imposta la riga in cui si trova il cursore come prossimo enunciato da eseguire. Usate questo comando per rieseguire degli enunciati o per saltare quelle che non volete eseguire.

Il menu Visualizza

Il menu Visualizza contiene un comando correlato al debug:

- Schermo output (F4) mostra i contenuti dello schermo di output. Usate questo comando mentre esaminate il programma per vederne i risultati sullo schermo di output. Per tornare nell'ambiente di modifica basta premere un tasto qualsiasi

Il menu Opzioni

Il menu Opzioni contiene un comando correlato al debug:

- Verifica sintassi è un comando interruttore che disattiva il controllore automatico di sintassi di QBasic, che è normalmente attivo.

In aggiunta a questi comandi si può usare il tasto F7 per eseguire a velocità piena la riga corrente e le righe comprese tra questa e il cursore. Questo tasto è una scorciatoia per l'impostazione di breakpoint con il tasto F9 e per l'esecuzione del programma dalla riga corrente al breakpoint col tasto F5.

CONTROLLO DELLE VARIABILI CON PRINT

Un altro utile strumento per il debug è un enunciato che abbiamo già visto, PRINT. Inserendo questi enunciati nel programma o usandoli nella Finestra Immediato si possono esaminare i contenuti delle variabili durante l'esecuzione di una programma (può essere utile visualizzare questi enunciati in un colore diverso per differenziarli dai risultati del programma). Una volta usati per il debug non scordate di togliere gli enunciati PRINT dal vostro programma.

L'esercizio seguente usa PRINT per trovare un errore logico contenuto nel programma.

Esercizio: Uso di PRINT per controllare una variabile che cambia

Il programma TROVABUG.BAS (figura 12-1) è un semplice indovinello che chiede all'utente di indovinare la dimensione in galloni del più grande frappè mai fatto (un record stabilito nell'Ohio nel 1988). Sebbene il programma venga eseguito senza errori di sintassi, non funziona in maniera corretta: indipendentemente dal numero inserito, risponde sempre che il numero è troppo basso. Riuscite a trovare l'errore logico?

Scrivete ed eseguite il programma TROVABUG.BAS.

Al momento dell'esecuzione avrete un output simile a questo:

```
Di quanti galloni è  il più grande frappé mai fatto?

Risposta: 100    troppo basso
Risposta: 174    troppo basso
Risposta: 1000   troppo basso
Risposta: 15000  troppo basso
Risposta: Ctrl-C
```

Indipendentemente dal valore specificato (compreso quello esatto, 174) il messaggio che appare è sempre lo stesso: *troppo basso*. Il programma è bloccato in un loop infinito e il solo modo per interromperlo è quello di premere Ctrl-C o Ctrl-Break.

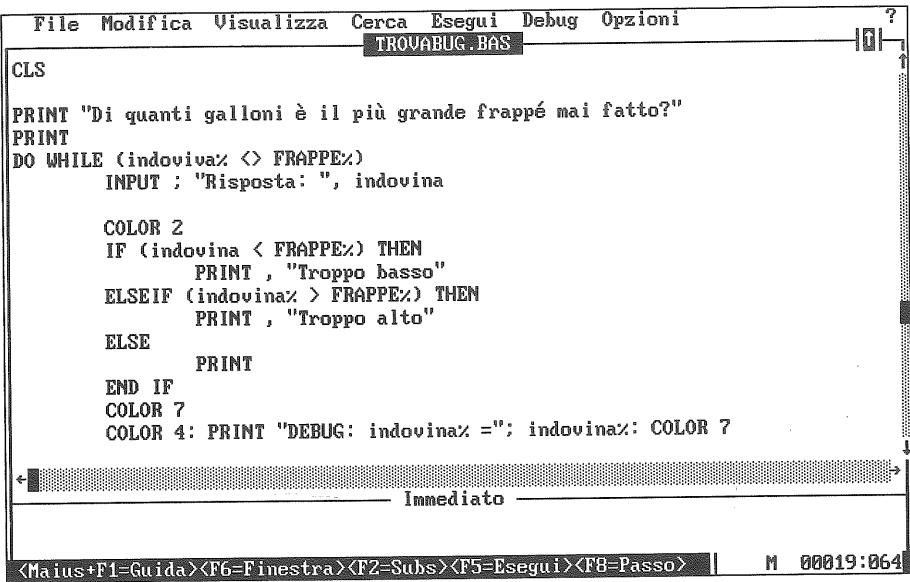
```
' TROVABUG.BAS
' Questo programma contiene un errore logico. Riuscite a trovarlo?

CONST FRAPPE% = 174    ' costante dei galloni del frappé più grande

CLS

PRINT "Di quanti galloni è il più grande frappé mai fatto?"
PRINT

DO WHILE (indovina% <> FRAPPE%) ' confronta la risposta il frappé più
                                ' grande
   INPUT ; "Risposta: ", indovina ' riceve la risposta dall'utente

   COLOR 2   ' imposta il colore verde per il suggerimento
   IF (indovina% < FRAPPE%) THEN   ' determina se la risposta
                        ' è troppo alta o bassa
      PRINT , "troppo basso"' e stampa il suggerimento appropriato
   ELSEIF (indovina% > FRAPPE%) THEN
      PRINT , "troppo alto"
   ELSE
      PRINT
   END IF
   COLOR 7   ' imposta il colore bianco predefinito
LOOP

PRINT
PRINT "Esatto! Un frappé alla fragola di"; FRAPPE%; "galloni è"
PRINT "stato fatto nell'Ohio nel 1988."
```

Figura 12-1. *Un programma indovinello contenente un errore logico.*

Proviamo a individuare l'errore usando l'enunciato PRINT per controllare il valore della variabile indovina% che cambia quando l'utente specifica un nuovo valore nel loop DO.

1. Inserite gli enunciati seguenti (che visualizzano il valore della variabile *indovina* in rosso) alla fine del loop DO, prima dell'enunciato LOOP:

```
COLOR 4: PRINT "DEBUG: indovina% ="; indovina%: COLOR 7
```

Dopo questa modifica il vostro schermo dovrebbe apparire come mostrato nella pagina successiva.

L'enunciato PRINT è circondato da due enunciati COLOR: il primo cambia il colore in primo piano in rosso, il secondo lo riporta al colore bianco predefinito. Questo secondo colore può essere utile per distinguere l'output del programma dall'output degli enunciati di debug. Se il vostro schermo non è in grado di visualizzare il colore potete omettere gli enunciati COLOR.

2. Eseguite nuovamente il programma e otterrete un output simile al seguente, completo di informazioni di debug:

```
Di quanti galloni è il più grande frappé mai fatto?

Risposta: 100     troppo basso
DEBUG:      indovina% = 0
Risposta: 1000    troppo basso
DEBUG:      indovina% = 0
Risposta: 174     troppo basso
Risposta: Ctrl-C
```

Il risultato dell'enunciato PRINT è interessante perché nostra che per qualche ragione la risposta non viene assegnata alla variabile *indovina%*. Ci sono due possibili spiegazioni: vi è qualcosa di sbagliato con l'enunciato INPUT o la variabile *indovina%* è stata modificata in qualche punto. Riuscite a trovare l'errore adesso?

Errori di programmazione comuni

I seguenti errori logici sono molto comuni, soprattutto in conseguenza di errori di digitazione o di progettazione.

- **Errate assegnazioni di tipo.** Assicuratevi che i valori dei dati non siano assegnati a variabili di tipo sbagliato; errate assegnazioni possono non essere riconosciute da QBasic causando problemi nell'esecuzione del programma. Controllate bene le assegnazioni contenute negli enunciati INPUT, INPUT# e READ.
- **Confusione tra nome e variabile.** Scrivete correttamente i nomi delle variabili e non omettete i caratteri di dichiarazione del tipo; fate attenzione a non confondere variabili locali e globali con lo stesso nome (ricordate che QBasic non controlla i nomi delle variabili.)
- **Confronto logico errato.** Controllate i confronti logici compiuti da operatori relazionali in enunciati IF, SELECT CASE e di loop. Usate i programmi di esempio o la Finestra Immediato per verificare che i confronti funzionino correttamente.
- **Loop infiniti.** Fate attenzione a questi tipi di loop; potete usare la Finestra Immediato per controllare le condizioni finali dei loop.
- **Output non chiaro.** Un uso sbagliato degli enunciati PRINT, PRINT USING, LOCATE e SCREEN possono produrre un output non chiaro o confuso. Assicuratevi che i programmi siano in grado di gestire qualunque tipo di dati inserito dall'utente.
- **Problemi con le matrici.** Non confondete il numero indice di una matrice con il valore in essa conservato. Per evitare il messaggio di errore *Indice inferiore fuori limite* non leggete o scrivete oltre la fine di una matrice.

Se esaminate l'enunciato INPUT scoprirete che il problema è nella variabile *indovina%*; infatti, nell'enunciato INPUT, la variabile indovina è priva del carattere di dichiarazione del tipo (%):

```
INPUT ; "Risposta: ", indovina
```

Perciò QBasic considera *indovina* e *indovina%* come variabili separate.

3. Correggete l'enunciato INPUT aggiungendo il simbolo % dopo *indovina:*


```
INPUT ; "Risposta: ", indovina%
```

4. Eseguite il programma per assicurarvi che venga eseguito correttamente e poi togliete gli enunciati di debug che avevate aggiunto in precedenza.

DEBUGGING DI UN PROGRAMMA IN DETTAGLIO

Per sperimentare il controllo e la correzione degli errori, in questo paragrafo mostreremo il debug del programma DESSERT.BAS che usa una matrice per registrare le torte preferite e le loro calorie.

DESSERT.BAS senza bug

DESSERT.BAS chiede all'utente il nome e il numero di torte da inserire; il numero inserito viene usato per dimensionare una matrice monodimensionale di valori stringa in cui vengono conservati i nomi delle torte. Un sottoprogramma aggiunge poi i dati alla matrice e fa la somma del numero di calorie inserite. Infine il programma stampa i contenuti della matrice e mostra le calorie totali in colore rosso lampeggiante.

L'output di DESSERT.BAS (se eseguito correttamente) dovrebbe essere:

```
Benvenuti nel programma DESSERT!

Scrivete il vostro nome: Lorenzo

Quanti nomi di torte vuoi inserire, Lorenzo? 2

Nome della torta: Crostata di ciliege
Calorie della torte: 125

Nome della torta: Torta al cioccolato
Calorie nella torta: 310

Questa è la lista delle torte preferite di Lorenzo:

Crostata di ciliege
Torta al cioccolato

La lista contiene un totale di 435 calorie!
```

Esercizio: Esecuzione del programma DESSERT.BAS

Il programma DESSERT.BAS (figura 12-2) contiene due errori logici. Provate a scriverlo esattamente come appare nella figura ed eseguitelo per scoprire gli errori.

```
' DESSERT.BAS
' Questo programma contiene due errori logici. Riuscite a trovarli?
    ' dichiara il sottoprogramma RiceviDati
DECLARE SUB RiceviDati (matrice$(), num%, totaleCalorie%)

CLS
    ' stampa il messaggio di benvenuto
PRINT "Benvenuti nel programma DESSERT!"
PRINT ' riceve il nome dell'utente
INPUT "Scrivete il vostro nome: ", nomeUtente$
PRINT ' riceve il numero di torte
PRINT "Quanti nomi di torte vuoi inserire, "; nomeUtente$;
INPUT num%
PRINT
DIM torte$(num%)   'dimensiona la matrice stringa
RiceviDati$(), num%, totaleCalorie%   ' chiama il sottoprogramma
PRINT "Questa è la lista delle torte preferite di "; nomeUtente$
PRINT
FOR i% = 1 TO num%
    PRINT "    "; torte$(i%)   ' stampa i contenuti della matrice
NEXT i%

PRINT ' stampa il numero di calorie totali
PRINT "La lista contiene un totale di";   ' numero in rosso lampeggiante
COLOR 20: PRINT totaleCalorie%; : COLOR 7
PRINT "calorie!"
END
SUB RiceviDati (matrice$(), num%, totaleCalorie%)
' Questo sottoprogramma riceve le informazioni sulle torte dall'utente.
FOR i% = 1 TO num% ' ripete un loop num% volte (valore passato dal
programma principale)
   INPUT "  Nome della torta: ", matrice$(num%)' riceve il nome torta
   INPUT "  Calorie nella torta: ", calorie%   ' riceve le calorie
   PRINT
   totaleCalorie% = calorie%' esegue il totale
NEXT i%

END SUB   'restituisce matrice% e totaleCalorie% al programma principale
```

Figura 12-2. *Un programma contenente due errori logici.*

L'output che otterrete sarà il seguente:

```
Benvenuti nel programma DESSERT!

Scrivete il vostro nome: Lorenzo

Quanti nomi di torte vuoi inserire, Lorenzo? 2

Nome della torta: Crostata di ciliege
Calorie della torte: 125

Nome della torta: Torta al cioccolato
Calorie nella torta: 310

Questa è la lista delle torte preferite di Lorenzo:


Torta al cioccolato

La lista contiene un totale di 310 calorie!
```

Si possono vedere subito due problemi: il programma mostra solo uno dei valori contenuti nella matrice *torte$* e non mostra il corretto valore delle calorie totali. Tenendo a mente queste due osservazioni si può iniziare a fare il debug del programma.

Il debug del programma DESSERT.BAS

La parte introduttiva del programma sembra a posto (mostra informazioni utili e ottiene i dati dall'utente). A questo punto possiamo inserire un breakpoint in mezzo al programma ed eseguire a piena velocità la prima sezione del programma.

Impostazione di un breakpoint

Impostare un breakpoint e poi eseguire il programma fino a quel punto è semplice:

1. Selezionate il comando Riavvia dal menu Esegui per preparare il programma all'esecuzione dall'inizio (l'enunciato CLS, che è la prima eseguibile nel programma, viene evidenziata).

2. Spostate il cursore in corrispondenza alla diciassettesima riga (quella che contiene l'enunciato DIM) e selezionate il comando Punto di interruzione dal menu Debug (F9).
3. Selezionate il comando Continua dal menu Esegui (F5) per eseguire il programma fino al breakpoint; inserite il vostro nome e il numero 2, in risposta alle richieste del programma. Questo si interromperà una volta raggiunto il breakpoint.
4. Selezionate il comando Punto di interruzione(F9) (il breakpoint non è più necessario).

Controllo del sottoprogramma RiceviDati

Cerchiamo ora di analizzare il sottoprogramma *RiceviDati* per capire che cosa c'è che non funziona.

1. Selezionate il comando Passo dal menu Debug (F8) per eseguire l'enunciato DIM che dimensiona la matrice *torte$*. Questa dovrebbe essere grande abbastanza per contenere tutti i nomi inseriti dall'utente (questo è un valore conosciuto in anticipo e conservato nella variabile *num%*). Notate che *num%* appare nell'enunciato DIM e che questo viene eseguito senza errori (nessun problema dunque, fino a questo punto).
2. Selezionate il comando Passo (F8) tre volte per entrare nel sottoprogramma *RiceviDati* ed eseguire i primi due enunciati. Scrivete *Crostata di ciliege* come prima risposta.
3. Date un'occhiata ai contenuti del sottoprogramma *RiceviDati* per vedere se tutto è in ordine (a volte un bug può saltare all'occhio facilmente). Un modo per esaminare il processo di input fino a questo punto è quello di inserire un enunciato PRINT nella Finestra Immediato; in questo caso l'enunciato PRINT ci serve per vedere se il primo nome viene correttamente collocato in corrispondenza al primo elemento di *matrice$*. Premete F6 per spostarvi nella Finestra Immediato e poi inserite questa riga:

```
COLOR 4: PRINT "DEBUG: matrice$(1) = "; matrice$(1): COLOR 7
```

I due punti vi consentono di inserire più di un enunciato su una riga. Con questa nuova riga le informazioni di debug vengono mostrate sullo schermo in rosso:

```
DEBUG: matrice$(1) =
```


matrice$(1), il primo elemento della matrice *matrice$*, dovrebbe contenere la stringa *Crostata di ciliege*, appena inserita, ma non lo fa; questo significa che ci dev'essere qualche errore nell'enunciato INPUT. In particolare ci dev'essere qualcosa di sbagliato che riguarda l'indice dell'assegnazione di matrice: questo dovrebbe infatti essere uguale a 1 la prima volta che il loop viene eseguito.

4. Premete un tasto qualsiasi per tornare nella Finestra Immediato.

Isolamento del primo errore logico

Abbiamo appena diagnosticato il primo errore logico. Vediamo ora di isolarne le cause esaminando più in dettaglio l'enunciato INPUT e il suo ruolo nel loop FOR. Una regola base per l'efficacia degli indici di matrice e dei loop è che se il loop contiene una variabile di calcolo (che registra il numero di loop fatti) e tutti gli elementi della matrice devono essere usati, la variabile di calcolo (o un suo derivato) dovrebbe essere usata come indice della matrice. Il sottoprogramma *RiceviDati* è corretto tranne che per l'indice della matrice; l'enunciato INPUT dovrebbe usare la variabile di calcolo *i%* come indice della matrice, al posto della variabile *num%*.

Correggete l'errore premendo F6 per tornare nella Finestra di digitazione e poi cambiando l'enunciato INPUT in questo modo:

```
INPUT "   Nome della torta: ", matrice$(i%)
```

Controllo della correzione

Dopo aver corretto l'errore logico si può controllare nuovamente il programma:

1. Lasciate il cursore sulla riga che contiene l'input per il nome della torta e selezionate il comando Imposta istruzione successiva dal menu Debug: questo comando permette di rieseguire l'enunciato INPUT per controllare la correzione (consentendo così un notevole risparmio di tempo).
2. Premete F8 per eseguire il nuovo enunciato INPUT e poi inserite *Crostata di ciliege* di nuovo. Per verificare che questo input venga collocato correttamente nel primo elemento di *matrice$* basta premere F6, per passare alla Finestra Immediato, spostare il cursore all'enunciato di debug PRINT e premere Invio per eseguirlo; questa volta lo schermo dovrebbe mostrare il corretto valore stringa, *Crostata di ciliege*. Premete un tasto

qualsiasi per tornare nella Finestra Immediato e poi F6 per passare alla Finestra di digitazione.

3. Proseguite il controllo del loop FOR spostando il cursore sulla riga contenente l'enunciato END SUB (l'ultima riga del sottoprogramma); selezionate il comando Punto di interruzione (F9) per impostare un breakpoint su questa riga, e poi selezionate il comando Analizza il flusso dal menu Debug. In questo modo è possibile vedere le rimanenti istruzioni del loop FOR a velocità rallentata. All'inizio del programma avevamo specificato due torte, per cui il loop dovrebbe essere eseguito due volte. Selezionate il comando Continua (F5) per eseguire il programma a velocità rallentata; durante l'esecuzione del loop verrà richiesto di inserire le informazioni.

4. Sembra che la prima correzione abbia avuto effetto (il programma ha fatto un loop e si è fermato al breakpoint senza errori). Selezionate il comando Punto di interruzione (F9) per togliere il breakpoint e il comando Analizza il flusso per eseguire il programma a velocità normale.

Ricerca del secondo bug

Dobbiamo trovare ancora un bug all'interno del nostro programma.

1. Premete F8 una volta per tornare nel programma principale e sette altre volte per visualizzare alcuni enunciati PRINT e i contenuti della matrice *torte$*, che sono stati trasferiti dal sottoprogramma *RiceviDati.*

2. Premete F4 per visualizzare l'attuale schermo di output: tutti gli elementi sono corretti! Si noti che lo schermo di output sta diventando un po' pieno a causa dei messaggi di debug e del continuo riavviamento del programma; non c'è nulla di cui preoccuparsi perché tutto ciò non apparirà nella versione finale del programma. Premete un tasto qualsiasi per tornare nella Finestra Immediato.

3. Premete F8 sette altre volte per eseguire gli enunciati che mostrano il calcolo finale delle calorie e terminano il programma. Premete F4 per vedere i risultati: il calcolo delle calorie è completamente sbagliato. Anche qui può essere utile usare un enunciato PRINT per controllare la variabile *calorie%*. Premete infine un tasto qualsiasi per tornare nella Finestra di digitazione.

4. Spostatevi nella Finestra Immediato, cambiate l'enunciato di debug in questo modo e premete Invio.

```
COLOR 4: PRINT "DEBUG: totaleCalorie% ="; totaleCalorie%: COLOR 7
```

L'enunciato PRINT mostra l'ultimo valore del calcolo delle calorie inserite. Cosa succede? Non sarebbe stato compito del sottoprogramma *RiceviDati* tenere un totale aggiornato del valore delle calorie?

Isolamento del secondo errore logico

Torniamo nel sottoprogramma *RiceviDati* per trovare la fonte dell'errore. Una prima cosa da fare è controllare che non vi siano errori ortografici (il nome di una variabile potrebbe essere stato scritto male); tornate nella Finestra di digitazione e usate il comando Trova dal menu Cerca per cercare la variabile *totaleCalorie%* (si noti che il comando trova è molto utile quando si lavora su programmi di grandi dimensioni). Il comando Trova individua la variabile *totaleCalorie%* nella lista dei parametri del sottoprogramma RiceviDati; selezionate il comando Repti trova dal menu Cerca (F3) più volte fino a completare la ricerca: *totaleCalorie%* appare sei volte (tre delle quali nel sottoprogramma).

Si può concludere che non vi è un errore di ortografia. Passiamo allora a controllare l'enunciato nel sottoprogramma *RiceviDati* che contiene il totale aggiornato delle calorie inserite dall'utente; premete F2 e selezionate *RiceviDati* per effettuare delle modifiche. Esaminate l'enunciato di assegnazione alla fine del loop FOR:

```
totaleCalorie% = calorie%
```

Quest'enunciato aggiorna in modo appropriato *totaleCalorie%* per tenere un totale aggiornato?

Correzione del secondo errore logico

La risposta è no! Come abbiamo visto nei capitoli precedenti, per tenere aggiornato un totale è necessario sommare il totale corrente e il successivo valore aggiunto oppure leggere e poi assegnare al totale corrente. La modifica da apportare per correggere questo errore è la seguente:

```
totaleCalorie% = totaleCalorie% + calorie%
```

Una volta apportati questi cambiamenti il programma potrà essere eseguito con successo. Controllatelo nuovamente dall'inizio alla fine per assicurarvi che sia completamente libero da bug.

Come evitare i bug

Dopo aver visto come controllare e correggere bug, vediamo ora alcuni suggerimenti in grado di aiutarvi a ridurre il rischio di errori nei programmi.

Pianificare attentamente

Prima di iniziare a scrivere un programma assicuratevi di sapere che cosa volete che faccia e come pensate di scriverlo in QBasic. Pensate ai vari algoritmi che il programma userà, come l'input e l'output verranno organizzati e come i dati verranno conservati e manipolati. Cominciate in modo semplice e affrontate gli aspetti complessi del programma in una seconda fase.

Lavorare un passo alla volta

Non cercate di scrivere un programma tutto in una volta ma createlo e controllatelo un pezzo alla volta; isolate funzioni diverse in modo da creare sottoprogrammi e funzioni appropriate.

Eseguire spesso il programma

Quando fate un cambiamento, eseguite il programma per assicurarvi che funzioni. In questo modo potete cogliere semplici errori di programmazione in anticipo.

Provare nuove idee nella Finestra Immediato

Usate la Finestra Immediato per controllare parti di codice prima di inserirle nel programma.

Controllare il programma rigorosamente

Controllate il programma in ogni sua parte e imparate a conoscere come risponderà a qualsiasi tipo di input; rispondere a queste domande vi può essere utile:

- Qual è il numero o la stringa più grande che il programma è in grado di gestire?
- Qual è il più piccolo?
- Cosa può causare l'interruzione del programma?
- Come si può evitare che un utente provochi un'interruzione del programma?
- L'utente sarà in grado di capire cosa fa il programma?

SOMMARIO

In questo capitolo abbiamo visto molti strumenti e tecniche di debug. Oggi vi sono molti strumenti disponibili per i programmatori e molti di questi sono disponibili in QBasic; ma a un programmatore non bastano gli strumenti per fare il debug dei suoi programmi (e peggio ancora di quelli di altri) ma è necessaria abitudine e abilità a pensare in modo logico e creativo, esaminando l'esecuzione dei programmi in tutti i suoi momenti. Evidentemente più si conoscono QBasic e le sue regole di sintassi, più facile sarà individuare gli errori di programmazione e trovare soluzioni adeguate.

DOMANDI ED ESERCIZI

1. Quali passi si dovrebbero seguire per isolare un bug in un programma?
2. Qual è la differenza tra un errore di sintassi e un errore run-time?
3. Se poteste disporre di soli due comandi del menu Debug, quali scegliereste?
4. Cosa fa il tasto funzione F7?
5. Qual è lo scopo del comando Imposta istruzione successiva?
6. Che tipo di errore logico è più difficile da individuare e correggere?
7. Il programma (ORSO.BAS), nella pagina successiva contiene due errori di sintassi; scrivetelo, trovate gli errori e correggeteli.

```
' ORSO.BAS
' Questo programma contiene due errori di sintassi.
CLS
DIMM orsi$(5)       ' dimensiona la matrice stringa
PRINT "Inserite i nomi dei vostri cinque orsi preferiti."
PRINT
FOR i% = 1 TO 5    ' riceve le cinque stringhe
      INPUT "Orso: ", orsi$(i%))
NET i%
PRINT
PRINT "Avete inserito questi orsi:"
PRINT
FOR i% = 1 TO 5    ' stampa le cinque stringhe
      PRINT orsi$(i%)
```

8. Il programma ERNOMI.BAS contiene due errori logici. Scrivete il programma e usate gli strumenti visti in questo capitolo per correggerli.

```
' ERNOMI.BAS
' Questo programma separa nomi e cognomi per poi stamparli.

CLS

PRINT "Scrivete il vostro nome e cognome in questo formato: ";
PRINT "Cognome, Nome"
PRINT

INPUT "Nome: ", nomeIntero$

virgolaPosto% = INSTR(1, nomeIntero$, ",")

IF (virgolaPosto% < 0) THEN
      Cognome$ = LEFT$(nomeIntero$, virgolaPosto% - 1)
      Nome$ = RIGHT$(nomeIntero$, LEN(nomeIntero$) - virgolaPosto% - 1)

      PRINT
      PRINT "Che bel nome! Piacere di conoscerti, ";
      PRINT Nome$; " "; Cognome$; "!"
ELSE
      PRINT
      PRINT "Il nome non è in formato Cognome, Nome."
END IF
```


CAPITOLO 13

Approfondire la conoscenza di QBasic

Congratulazioni! Avete completato questo libro che aveva lo scopo di introdurvi alla programmazione in QBasic fornendovi i primi strumenti per cominciare a programmare. Speriamo che questo non sia che un punto di partenza verso un'esplorazione e una conoscenza ancor più approfondita di QBasic. In questo capitolo viene data una breve descrizione degli argomenti trattati in questo libro e un'introduzione ad alcune tematiche che meritano ulteriori approfondimenti da parte di chi sia interessato a imparare QBasic più in profondità. Inoltre verrà data una breve descrizione del Microsoft QuickBasic Compiler, una versione avanzata di QBasic, disponibile per chi si senta pronto a fare un ulteriore salto qualitativo.

COSA AVETE IMPARATO

Pensate per un attimo ai progressi che avete compiuto leggendo questo libro e seguendo gli esercizi via via proposti. Avete visto, tra le altre cose, come QBasic ragiona e come funziona un computer; avete appreso la struttura di un programma in QBasic e come crearne uno usando i comandi di menu. Avete imparato come conservare informazioni in un luogo per poi usarle in un altro e come ricevere un input dall'utente per poi usarlo nel programma. Avete visto come programmare in modo efficiente, usando i loop per compiti ripetitivi e dichiarando costanti per valori che non cambiano mai; avete anche imparato a progettare un programma e a correggerlo quando qualcosa non va. Infine avete affrontato i problemi inerenti all'uso della grafica e del suono per conferire ai vostri programmi un aspetto più piacevole e funzionale. Avete poi creato dei veri e propri programmi servendovi di tutte queste tecniche.

Nello sviluppare i vostri programmi non cercate di adottare sempre soluzioni nuove, ma usate, quando potete, parti con funzionalità generale. Raccogliete i listati di tutti i programmi che scrivete, perché questi possono rappresentare un'utile fonte di informazioni.

COSA DOVETE APPROFONDIRE

Le modalità di gestione di dati, input e output, e azioni ripetitive, viste in questo libro, rappresentano le fondamenta dei programmi che scriverete in futuro. Ma ci sono molte altre cose che si possono imparare su QBasic e ve ne accorgerete proprio lavorando con questo linguaggio; incontrerete situazioni che aprono tutta una serie di nuove domande: come lavora QBasic col mouse? Come posso trarre vantaggio dalle funzionalità delle stampanti? Come

funzionano i file ad accesso casuale? Come posso usare QBasic insieme ad altri linguaggi di programmazione?

Tutte queste domande trovano risposta nei moltissimi libri oggi in commercio.

ALTRE VERSIONI DI BASIC

MS-DOS QBasic non è il solo pacchetto per la programmazione in Basic della Microsoft Corporation. In termini di potenza rappresenta un prodotto di livello iniziale nella famiglia dei prodotti Microsoft. Se volete avventurarvi in prodotti più complessi non avete che l'imbarazzo della scelta.

Microsoft QuickBasic Compiler

Questa versione completa di MS-DOS QBasic contiene più di ventiquattro opzioni di menu che supportano debug di alto livello, gestione di file, procedure e programmazione modulare. Il maggior vantaggio di questo prodotto è che permette di creare librerie e file eseguibili *stand-alone* (autonomi), rendendolo concorrenziale con tutti gli altri pacchetti di programmazione avanzata. QuickBasic Compiler è il passo più logico per chi voglia continuare a programmare in Basic.

Microsoft QuickBasic per Apple Macintosh

Questo compilatore QBasic è dedicato alla famiglia di personal computer Apple Macintosh. Questa versione è simile a quella per personal computer MS-DOS ma è studiata per supportare l'ambiente operativo, basato su menu e finestre, proprio del Macintosh.

Microsoft Basic Professional Development System

È un compilatore avanzato per programmatori professionali. Include supporto per il sistema operativo OS/2 e per progetti di programmazione di grandi dimensioni; comprende una copia del Microsoft QuickBasic Compiler.

MICROSOFT QUICKBASIC COMPILER

MS-DOS QBasic è un prodotto completo e utile per imparare a programmare in Basic e per sviluppare semplici applicazioni. Se però avete bisogno di funzionalità più complesse, potete orientarvi verso il Microsoft QuickBasic Compiler, che presenta molti vantaggi rispetto a QBasic:

- Ha la capacità di creare file eseguibili *stand-alone* (file con estensione.EXE).
- Può creare librerie per procedure usate spesso.
- Dispone di tecniche avanzate di debug.
- Supporta programmi con più moduli.
- Permette un maggior controllo sull'ambiente di programmazione.
- Contiene un numero maggiore di opzioni di menu.

Inoltre tutte queste funzionalità sono integrate nel tipico ambiente di programmazione di QBasic; in questo modo tutto ciò che avete imparato fino a questo momento non va perduto. Inoltre, il linguaggio di programmazione Basic è identico in entrambi i prodotti, per cui tutti i programmi scritti in QBasic potranno essere eseguiti senza problemi.

SOMMARIO

In questo capitolo abbiamo evidenziato i concetti appresi fino ad ora. Ci auguriamo che questo libro vi spinga a esplorare e a imparare ancor di più il linguaggio Basic. Indipendentemente dalle ragioni che vi hanno spinto a comprare questo libro, le cose che avete imparato non potranno che servirvi. Avete fatto il primo passo in un nuovo mondo: andatene fieri! E programmate, programmate, programmate!

APPENDICE A

Uso dei menu e delle opzioni di QBasic

Questa appendice rappresenta una veloce guida di riferimento all'ambiente di programmazione di QBasic basato sull'uso dei menu. Gli aspetti che verranno affrontati sono i seguenti:

- Selezione di menu e comandi.
- Uso di finestre di dialogo.
- Uso dei comandi di modifica da tastiera.
- Selezione di testo.
- Uso del sistema di aiuto on-line.
- Stampa dei programmi o di parte del sistema di aiuto.
- Cambiamento dei colori dello schermo.
- Uso delle opzioni di avvio.

Per informazioni più approfondite si può sempre fare riferimento al sistema di aiuto on line di QBasic.

SELEZIONE DI MENU E DI COMANDI

In QBasic vi sono otto menu a scorrimento (detti anche a tendina) che contengono comandi per la gestione dei file, l'uso della guida, la modifica, l'esecuzione e il debugging dei programmi. È possibile selezionare menu e comandi con il mouse o con la tastiera.

Con il mouse...

1. Posizionare il puntatore del mouse sul nome del menu da selezionare e fare clic con il pulsante sinistro per aprirlo.

2. Spostare il puntatore sul comando desiderato e fare clic nuovamente con il pulsante sinistro.

Con la tastiera...

1. Premere il tasto Alt per attivare la barra dei menu.

2. Premere il tasto corrispondente alla prima lettera del nome del menu da selezionare.

3. Usare i tasti di direzione per evidenziare il comando da eseguire e premere Invio (o semplicemente premere la lettera evidenziata nel nome del comando).

USO DI FINESTRE DI DIALOGO

Quando QBasic necessita di informazioni addizionali per un comando apre una finestra di dialogo. La figura A-1 mostra la finestra di dialogo che appare quando si seleziona il comando Apri dal menu File (si noti che i contenuti potranno essere diversi).

Una tipica finestra di dialogo contiene vari campi:

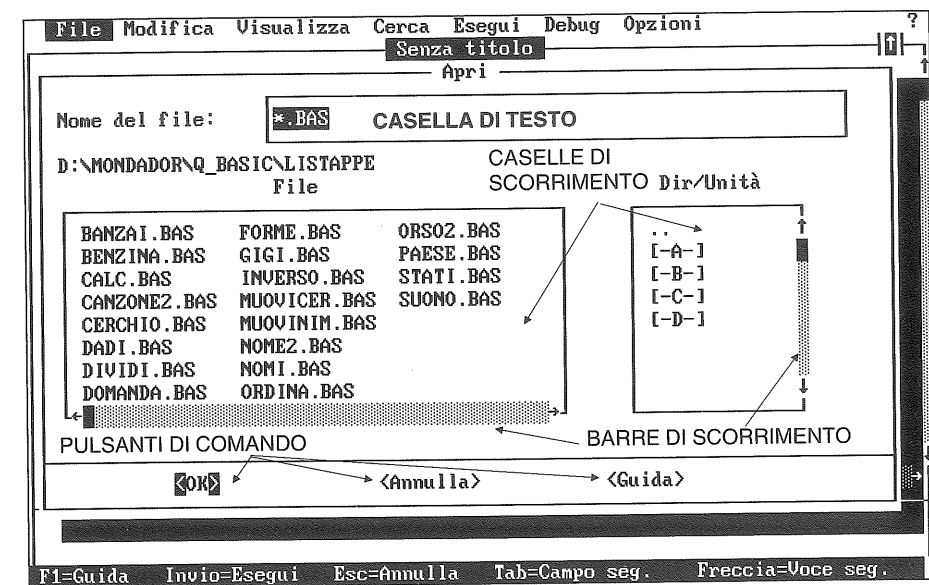



Figura A-1. *Gli elementi della finestra di dialogo Apri* .

- **Caselle di testo** per ricevere informazioni dalla tastiera, come il nome o il percorso di ricerca di un file.
- **Caselle a scorrimento** per la selezione di un elemento (come il nome di un file, di una directory o un colore) tra diverse opzioni. Le *barre di*

scorrimento permettono di vedere anche quelle informazioni che non si trovano nelle caselle.

- **Caselle di controllo** per indicare se un elemento è attivo o meno, attraverso la presenza di una X tra parentesi quadre.
- **Pulsanti opzione** sono gruppi di pulsanti tra cui è possibile effettuare una sola selezione.
- **Pulsanti di comando** per controllare l'operatività di una finestra di dialogo. Questi possono essere usati per eseguire un comando, cancellarlo o ottenere l'aiuto in linea.

I campi di una finestra di dialogo possono essere selezionati con il mouse o con la tastiera:

Con il mouse...

1. Fare clic sul campo da modificare.
2. Digitare le informazioni (solo nelle caselle di testo) o fare clic sull'elemento desiderato.
3. Fare clic sul pulsante OK per eseguire il comando, sul pulsante Annulla per cancellarlo o sul pulsante Guida per ottenere la guida on-line.

Con la tastiera...

1. Premere il tasto Tab per spostarsi all'interno della finestra di dialogo.
2. Digitare le informazioni (solo nelle caselle di testo) o usare i tasti di direzione per selezionare l'elemento desiderato.
3. Premere Invio per eseguire il comando o Esc per cancellarlo.

I seguenti tasti possono essere usati all'interno di una finestra di dialogo:

Tasto(i)	Funzione(i)
Tab	Sposta al campo successivo
Maiusc-Tab	Sposta al campo precedente
Tasti di direzione	Seleziona un elemento all'interno dell'elenco delle voci; seleziona un pulsante opzione; attiva\disattiva una casella di controllo
Pag↑/Pag↓	Scorre l'elenco delle voci di diversi elementi alla volta.
Barra spaziatrice	Attiva/disattiva una casella di controllo; esegue un pulsante di comando
F1	Mostra le istruzioni per il completamento di una finestra di dialogo
Invio	Esegue un comando
Esc	Cancella un comando

Barre di scorrimento

Le barre di scorrimento permettono di vedere più testo di quanto non sia visibile in una finestra. Usate le barre di scorrimento per visualizzare il testo nascosto.

Uso delle barre di scorrimento col mouse

Per usare le barre di scorrimento col mouse basta fare clic su una delle frecce in esse contenute, così facendo vengono visualizzate righe o colonne di informazioni non visibili precedentemente. Facendo clic sulla barra stessa ci si può spostare velocemente nella lista; per spostarsi in una posizione specifica è anche possibile trascinare la barra di scorrimento.

Uso delle barre di scorrimento con la tastiera

Per usare le barre di scorrimento con la tastiera basta premere i tasti di direzione evidenziando così le righe di informazioni nascoste (Pag ↓ e Pag↑ permettono di scorrere molte righe in una sola volta).

Premendo il tasto corrispondente a una lettera è possibile spostarsi agli elementi che iniziano con quella lettera.

EQUIVALENTI DI TASTIERA DEI COMANDI DI MENU

QBasic fornisce scorciatoie di tastiera per la maggior parte dei comandi di menu e per alcuni comandi che non si trovano all'interno di essi, come descritto nella seguente tabella.

Tasto(i)	Funzione	Menu/Comando
F1	Mostra la guida specifica	Guida/Argomento
Maiusc-F1	Mostra la guida generale	Guida/Guida
F2	Mostra l'elenco delle procedure	Visualizza/SUBs
Maiusc-F2	Mostra la procedura successiva	Nessuno
Ctrl-F2	Mostra la procedura precedente	Nessuno
F3	Trova la successiva stringa da cercare	Cerca/Ripeti trova
F4	Mostra lo schermo di output	Visualizza/Schermo output
F5	Continua l'esecuzione del programma	Esegui/Continua
Maiusc-F5	Esegue un programma dall'inizio	Esegui/Avvia
F6	Attiva la finestra successiva	Nessuno
Maiusc-F6	Attiva la finestra precedente	Nessuno
F7	Esegue il programma al cursore	Nessuno
F8	Esegue l'enunciato successivo	Debug/Passo
F9	Attiva/disattiva il breakpoint	Debug/Punto di interruzione
F10	Esegue l'enunciato successivo (salta la procedura)	Debug/Procedura passo
Ctrl-F10	Commuta da finestre a pieno schermo e finestre multiple	Nessuno

COMANDI DI TASTIERA PER LA MODIFICA

Oltre ai comandi di modifica del menu Modifica, QBasic dispone di diversi comandi di modifica da tastiera, molti dei quali hanno degli equivalenti in WordStar, come mostra questa tabella.

Tasto(i)	Tasti WordStar	Funzione
←	Ctrl-S	Sposta un carattere a sinistra
→	Ctrl-D	Sposta un carattere a destra
Ctrl-←	Ctrl-A	Sposta una parola a sinistra
Ctrl-→	Ctrl-F	Sposta una parola a destra
Ctrl-E		Sposta una riga in su
Ctrl-X		Sposta una riga in giù
Home	Ctrl-QS	Sposta al primo carattere della riga

Tasto(i)	Tasti WordStar	Funzione
Fine	Ctrl-QD	Sposta all'ultimo carattere nella riga
Ctrl-Invio	Ctrl-J	Sposta all'inizio della riga successiva
Ctrl-Home	Ctrl-QR	Sposta all'inizio della procedura
Ctrl-Fine	Ctrl-QC	Sposta alla fine della procedura
Ins	Ctrl-V	Attiva/disattiva la modalità inserimento
Maiusc-Ins	Nessuno	Incolla i contenuti degli Appunti
Maiusc-Canc	Nessuno	Taglia il testo selezionato (lo pone negli Appunti)
Nessuno	Ctrl-Y	Taglia la riga corrente (la pone negli Appunti)
Nessuno	Ctrl-QY	Taglia dal cursore alla fine della riga (lo pone negli Appunti).
Canc	Ctrl-G	Cancella il testo o il carattere selezionato
Nessuno	Ctrl-T	Cancella la parola dalla posizione del cursore
Maiusc-Tab	Nessuno	Cancella un livello di rientro per le righe selezionate
Ctrl-Ins	Nessuno	Copia il testo selezionato negli Appunti
PAG↑	Ctrl-R	Scorre un pagina verso l'alto
PAG↓	Ctrl-C	Scorre una pagina verso il basso
Ctrl-P	Nessuno	Inserisce il carattere di controllo (digitate Ctrl-P e poi il carattere di controllo desiderato)

SELEZIONE DI TESTO

Selezione è il processo di evidenziazione di un blocco di testo con la tastiera o col mouse; il testo selezionato può essere usato con molti comandi di QBasic, inclusi i seguenti.

Menu/Comando	Tasto(i)	Funzione
Modifica/Taglia	Maiusc-Canc	Cancella il testo selezionato (lo pone negli Appunti)
Modifica/Copia	Ctrl-Ins	Copia il testo selezionato negli Appunti
Modifica/Incolla	Maiusc-Ins	Incolla i contenuti degli Appunti
File/Stampa	Nessuno	Stampa il testo selezionato
Cerca/Trova	Nessuno	Cerca il testo selezionato
Cerca/Cambia	Nessuno	Cerca e cambia il testo selezionato
Modifica/New SUB	Nessuno	Usa il testo selezionato come nome per un nuovo sottoprogramma

Menu/Comando	Tasto(i)	Funzione
Modifica/Nuova FUNCTION	Nessuno	Usa il testo selezionato come nome per una nuova funzione

NOTA: Fate molta attenzione quando lavorate con testo selezionato perché se digitate un carattere (anche uno spazio) mentre una riga di testo è selezionata questa viene sostituita (e persa definitivamente). Per cancellare una selezione (cioè per "deselezionarla") basta fare clic col pulsante sinistro del mouse o premere un tasto di direzione.

La selezione del testo può avvenire sia con il mouse che con la tastiera.

Con il mouse...

1. Posizionate il puntatore sul primo carattere da selezionare (esso dipende dalla direzione in cui spostate il mouse — si veda la tabella successiva).
2. Tenere premuto il tasto sinistro del mouse.
3. Spostare il puntatore all'ultimo carattere da selezionare.
4. Rilasciate il pulsante del mouse.

Gli spostamenti del mouse producono i seguenti effetti al momento della selezione di testo:

Spostamento	Effetto
Doppio clic	Seleziona la parola su cui è posizionato il puntatore del mouse
A destra di una colonna	Seleziona il carattere su cui è posizionato il puntatore del mouse
A sinistra di una colonna	Seleziona il carattere a sinistra del puntatore del mouse
Su di una riga	Seleziona la riga su cui è posizionato il puntatore e quella sopra*
Giù di una riga	Seleziona la riga su cui è posizionato il puntatore e quella sotto**

** Se il puntatore si trova nella prima colonna viene selezionata solo la riga superiore al punto d'inizio.*
*** Se il puntatore si trova nella prima colonna viene selezionata solo la riga contenente il punto di inizio.*

Con la tastiera...

1. Usare i tasti di direzione per spostare il cursore sul primo carattere da selezionare.
2. Tenere premuto il tasto Maiusc.
3. Usare i tasti di direzione per spostare il cursore sull'ultimo carattere da selezionare.
4. Rilasciare il tasto Maiusc.

I tasti di direzione producono i seguenti effetti al momento della selezione di testo:

Tasto(i) di direzione	Effetto
→	Seleziona il carattere su cui si trova il cursore
←	Seleziona il carattere alla sinistra del cursore
	Seleziona la riga su cui si trova il cursore e quella sopra*
	Seleziona la riga su cui si trova il cursore e quella sotto**
Pag↑	Seleziona la riga su cui si trova il cursore e tutta la schermata superiore
Pag↓	Seleziona la riga su cui si trova il cursore e tutta la schermata inferiore
Home	Seleziona dal carattere alla sinistra del cursore all'inizio di riga
Fine	Seleziona dal cursore alla fine della riga
Ctrl-Home	Seleziona dalla riga del cursore all'inizio della procedura
Ctrl-Fine	Seleziona dalla riga del cursore alla fine della procedura

** Se il cursore inizia nella prima colonna viene selezionata solo la riga superiore al punto d'inizio.*
*** Se il cursore inizia nella prima colonna viene selezionata solo la riga contenente il punto di inizio.*

UTILIZZO DELLA GUIDA

QBasic dispone di una documentazione in linea per parole chiave e altri elementi del linguaggio, per comandi, messaggi di errore e altri argomenti di programmazione. La guida in linea può essere attivata in tre modi:

- Per aspetti specifici basta porre il cursore sull'elemento (parole chiave etc.) per il quale si vuole aprire la documentazione e premere il tasto F1. Se si usa il mouse basta posizionare il puntatore sull'elemento e fare clic con il pulsante destro.

- Per un comando di menu o un messaggio di errore basta premere F1 quando il comando è evidenziato o quando il messaggio appare sullo schermo.
- Per argomenti generali di programmazione basta premere Alt-H per aprire il menu Guida e poi scegliere uno dei seguenti comandi:
  - *Indice* per vedere un elenco alfabetico di parole chiave e altri argomenti su QBasic
  - *Sommario* per vedere una lista di argomenti generali di programmazione come l'uso di QBasic, gli elementi del linguaggio e le informazioni tecniche
  - *Argomenti* per vedere la documentazione in linea della parola chiave su cui si trova il cursore
  - *Informazioni su* per avere informazioni sul sistema di aiuto in linea di QBasic.

La guida di QBasic è fatta di elementi interconnessi; ogni finestra di dialogo contiene collegamenti in *ipertesto* che rimandano ad altri argomenti della guida. Per selezionare uno di questi collegamenti con il mouse, basta fare doppio clic su di esso; con la tastiera, premere Tab per posizionare il cursore su di esso e poi premere Invio.

Se il testo della finestra di Guida è troppo esteso per essere visualizzato completamente, si possono usare i tasti Pag↓ e Pag↑ per scorrerlo. Per alternare tra la Guida, la Finestra di digitazione e la Finestra Immediato si usa il tasto F6.

Per uscire dalla Guida basta premere Esc.

NOTA: Se il vostro computer emette un suono quando premete F1 vuol dire che non esiste alcuna documentazione per la parola su cui il cursore si trova.

Copia di programmi dal sistema di guida

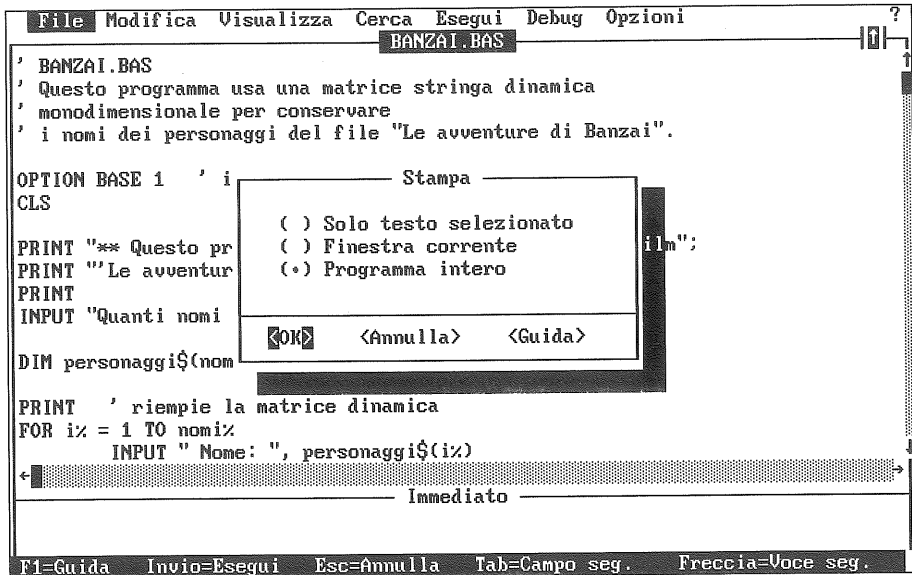
Copiare gli esempi di codice presenti nella guida all'interno dei vostri programmi è molto semplice:

1. Selezionare il codice da copiare.
2. Scegliere Copia dal menu Modifica.

3. Premere Esc per chiudere la finestra della Guida.
4. Spostare il cursore dove si intende copiare il codice.
5. Selezionare Incolla dal menu Modifica.

STAMPA

Se si dispone di una stampante è possibile copiare in tutto o in parte i programmi creati con il comando Stampa dal menu File.

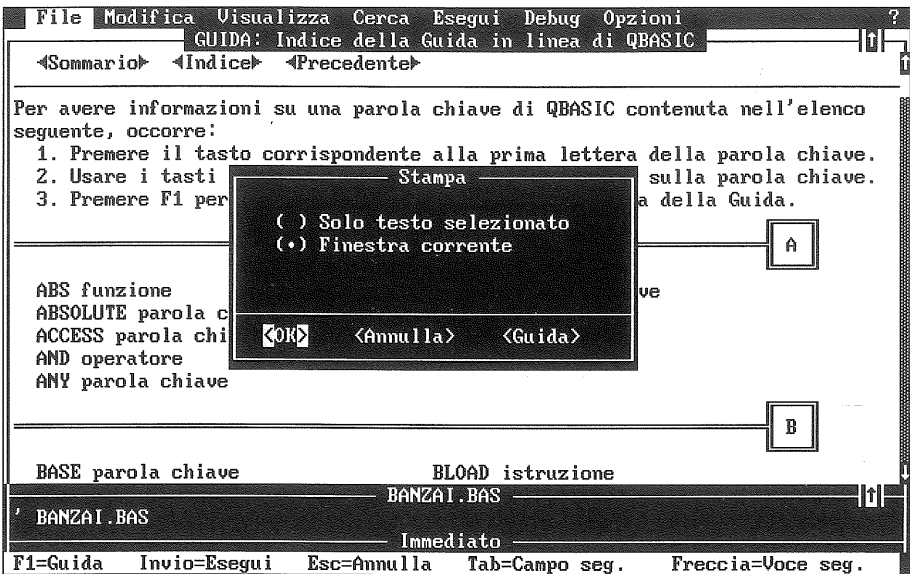
1. Selezionare il comando Stampa, che apre la seguente finestra di dialogo:

2. Selezionare l'opzione desiderata:
   - *Solo testo selezionato* stampa solo il testo selezionato (per usare questa opzione è necessario selezionare prima il testo che si vuole stampare).
   - *Finestra corrente* stampa solo i contenuti della finestra attiva (ciò è utile se si vuole stampare il programma principale, una procedura o i contenuti della Finestra Immediato.

- *Programma intero* stampa l'intero programma (compresi i sottoprogrammi e le funzioni).

3. Premere Invio o fare clic su *OK* per eseguire il comando Stampa.

Stampa di parti del sistema di guida

È anche possibile usare il comando Stampa per stampare sezioni della Guida in linea:

1. Individuare all'interno della Guida le parti da stampare.
2. Selezionare il comando Stampa che apre questa finestra di dialogo:

3. Selezionare l'opzione desiderata:
   - *Solo testo selezionato* stampa solo il testo selezionato (per usare questa opzione è necessario selezionare prima il testo che si vuole stampare).
   - *Finestra corrente* stampa un'intera voce della Guida.
4. Premere Invio o fare clic su *OK* per eseguire il comando Stampa.

CAMBIAMENTO DEI COLORI DELLO SCHERMO

QBasic permette di cambiare i colori dello sfondo e quelli in primo piano e di modificare altre caratteristiche dell'ambiente di programmazione per adeguarle alle preferenze personali. L'adattatore video e il monitor a disposizione determinano i colori utilizzabili.

1. Selezionate il comando Schermo dal menu Opzioni. La figura A-2 mostra la finestra di dialogo di questo comando; essa permette di cambiare i colori di fondo e di primo piano per tre elementi di testo sullo schermo:
   - *Testo normale* per cambiare i colori del testo regolare
   - *Istruzione corrente* per cambiare i colori dell'enunciato successivo che dev'essere eseguito
   - *Righe punto interruz.* per cambiare i colori delle righe di breakpoint impostate col comando Punto di interruzione dal menu Debug.

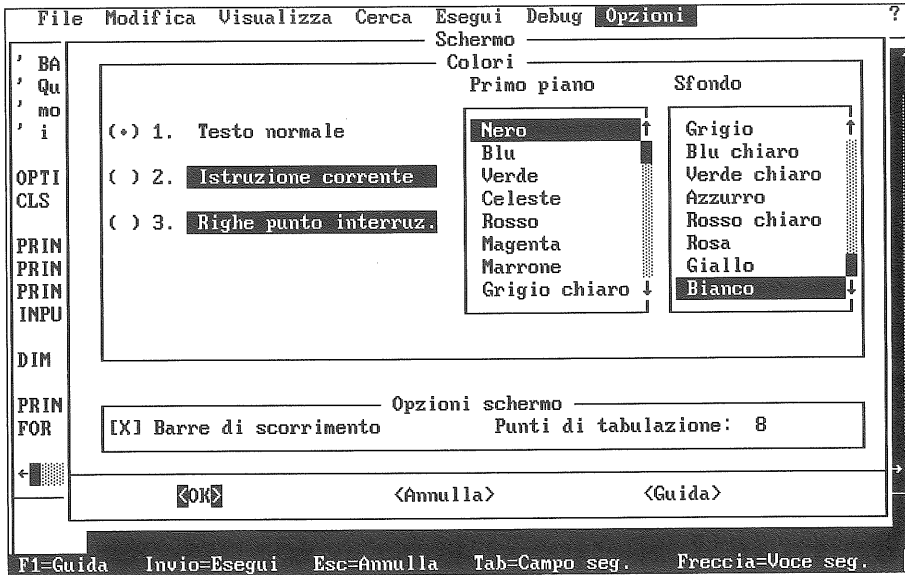
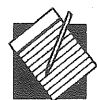



Figura A-2. *La finestra di dialogo del comando Schermo.*

2. Selezionare gli elementi di testo a cui volete cambiare colore.
3. Selezionare i colori per lo sfondo e per il primo piano.
4. Premere Invio o fate clic su *OK* per eseguire il comando Schermo.

NOTA: Se si scelgono dei colori diversi da quelli predefiniti si evitino colori troppo accesi e contrasti troppo forti per non stancare la vista.

La parte della finestra di dialogo intitolata *Opzioni Schermo* dà la possibilità di rimuovere le barre di scorrimento dallo schermo e di impostare la spaziatura delle tabulazioni. Togliendo le barre di scorrimento si ha la possibilità di vedere una parte più ampia del programma in una sola volta; impostando un valore per la spaziatura delle tabulazioni inferiore a 8 potete far stare una maggior quantità di codice sullo schermo e in stampa (ciò vale solo se siete abituati a rientrare le righe di codice).

OPZIONI DI AVVIAMENTO

QBasic è in grado di eseguire alcune funzioni direttamente al momento in cui viene avviato; per far ciò basta aggiungere delle informazioni addizionali (righe di comando) al comando di avviamento standard. Una lista parziale di queste righe di comando viene mostrata nella tabella seguente (per una lista completa potete fare riferimento all'aiuto in linea).

Opzione	Descrizione	Esempio
program	QBasic carica il programma indicato	*qbasic reversi*
/run program	QBasic carica ed esegue il programma indicato prima di mostrare il codice	*qbasic /run reversi*
/b	Esegue QBasic in modalità bianco e nero	*qbasic /b*
/ed	Esegue QBasic in modalità documento (identico al programma MS-DOS EDIT)	*qbasic /ed*
/h	Esegue QBasic con il massimo numero di righe per schermo ammesse dallo schermo (di solito 25, 43 o 50)	*qbasic /h*

APPENDICE B

Set di caratteri esteso ASCII e IBM

Le tabelle B-1 e B-2 mostrano i 256 caratteri disponibili sulla maggior parte dei personal computer basati su MS-DOS. La tabella B-1 mostra il set di caratteri ASCII e la B-2 il set di caratteri estesi IBM. È possibile vedere la maggior parte di questi caratteri con l'enunciato PRINT e confrontarle usando gli operatori relazionali di QBasic. Il capitolo 9 descrive le tecniche di visualizzazione e confronto in dettaglio.

Tabella B-1. *Il set di caratteri ASCII (codici 0-127)*

Carattere	ASCII	Carattere	ASCII	Carattere	ASCII	Carattere	ASCII
	0	<space>	32	@	64	`	96
☺	1	!	33	A	65	a	97
☻	2	"	34	B	66	b	98
♥	3	#	35	C	67	c	99
♦	4	$	36	D	68	d	100
♣	5	%	37	E	69	e	101
♠	6	&	38	F	70	f	102
•	7	'	39	G	71	g	103
◘	8	(	40	H	72	h	104
○	9	)	41	I	73	i	105
◙	10	*	42	J	74	j	106
♂	11	+	43	K	75	k	107
♀	12	,	44	L	76	l	108
♪	13	-	45	M	77	m	109
♫	14	.	46	N	78	n	110
☼	15	/	47	O	79	o	111
►	16	0	48	P	80	p	112
◄	17	1	49	Q	81	q	113
↕	18	2	50	R	82	r	114
‼	19	3	51	S	83	s	115
¶	20	4	52	T	84	t	116
§	21	5	53	U	85	u	117
▬	22	6	54	V	86	v	118
↨	23	7	55	W	87	w	119
↑	24	8	56	X	88	x	120
↓	25	9	57	Y	89	y	121
→	26	:	58	Z	90	z	122
←	27	;	59	[	91	{	123
∟	28	<	60	\	92	¦	124
↔	29	=	61	]	93	}	125
▲	30	>	62	^	94	~	126
▼	31	?	63	_	95	△	127

Tabella B-2. *Il set di caratteri estesi IBM (codici 128-255)*

Carattere	ASCII	Carattere	ASCII	Carattere	ASCII	Carattere	ASCII
Ç	128	á	160	@	192	a	224
ü	129	í	161	A	193	b	225
é	130	ó	162	B	194	G	226
â	131	ú	163	C	195	p	227
ä	132	ñ	164	D	196	(	228
à	133	Ñ	165	E	197	s	229
å	134	ª	166	F	198	m	230
ç	135	º	167	G	199	t	231
ê	136	¿	168	H	200	F	232
ë	137	⌐	169	I·	201	U	233
è	138	¬	170	J	202	V	234
ï	139	½	171	K	203	d	235
î	140	¼	172	L	204	`	236
ì	141	¡	173	M	205	f	237
Ä	142	«	174	N	206	e	238
Å	143	»	175	O	207	∩	239
É	144	4	176	P	208	[	240
æ	145	7	177	Q	209	6	241
Æ	146	9	178	R	210	≥	242
ô	147	3	179	S	211	≤	243
ö	148	4	180	T	212	⌠	244
ò	149	5	181	U	213	⌡	245
û	150	6	182	V	214	4	246
ù	151	7	183	W	215	'	247
ÿ	152	8	184	X	216	°	248
Ö	153	9	185	Y	217	•	249
Ü	154	:	186	Z	218	·	250
¢	155	;	187	[	219	√	251
£	156	<	188	\	220	h	252
¥	157	=	189	]	221	²	253
₧	158	>	190	^	222	▪	254
ƒ	159	?	191	_	223		255

Alcuni caratteri ASCII (da 0 a 32 o 127) vengono detti caratteri di controllo. Essi possono essere usati per eseguire compiti speciali, come l'avanzamento della carta nella stampante. La tabella B-3 elenca i caratteri di controllo usati più di frequente.

Tabella B-3. *Caratteri di controllo ASCII più usati*

Codice ASCII	Carattere di controllo	Descrizione
7	Campana	Suona il bip del computer
8	Backspace	Fa retrocedere di una posizione un carattere sullo schermo
9	Tab	Sposta a destra di una tabulazione
10	Alimentaz. riga	Avanza la stampante di una riga
12	Alimentaz. carta	Avanza la stampante di una pagina
13	Ritorno unitario	Avanza il cursore alla riga successiva

APPENDICE C

Enunciati e funzioni di QBasic

In questa appendice potete trovare il nome di tutte le funzioni e gli enunciati disponibili in QBasic. Per vedere una documentazione completa su ciascuno di questi elementi si può fare riferimento all'aiuto in linea.

AFFERMAZIONI

BEEP
BLOAD
BSAVE
CALL
CALL ABSOLUTE
CHAIN
CHDIR
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COM
COMMON
CONST
DATA
DATE$
DECLARE
DEF
DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR
DIM
RANDOMIZE
READ
REDIM
REM
RESET
RESTORE
RESUME
DO...LOOP
DRAW
END
ENVIRON
ERASE
ERROR
EXIT
FIELD
FILES
FOR...NEXT
FUNCTION
GET
GOSUB
GOTO
IF...THEN...ELSE
INPUT
INPUT$NO
IOCTL
KEY
KILL
LET
LINE
LINE INPUT
LINE INPUT$NO
LOCATE
SEEK
SELECT CASE
SHARED
SHELL
SLEEP
SOUND
STATIC
LOCK...UNLOCK
LPRINT
LPRINT USING
LSET
MID$
MKDIR
NAME
ON ERROR
ON
OPEN
OPTION BASE
OUT
PAINT
PALETTE
PCOPY
PEN
PLAY
PMAP
POKE
PRESET
PRINT
PRINT USING
PRINT$NO USING
PSET
PUT
TIME$
TIMER
TROFF
TRON
TYPE
UNLOCK
VIEW

RETURN
RMDIR
RSET
RUN
SCREEN
STOP
STRIG
SUB
SWAP
SYSTEM
WAIT
WHILE...WEND
WIDTH
WINDOW
WRITE

FUNZIONI

ABS
ASC
ATN
CDBL
CHR$
CINT
CLNG
COS
CSNG
CSRLIN
CVD
CVDMBF
CVI
CVL
CVS
CVSMBF
MKL$
MKSMBF$
MKS$
OCT$
PEEK
PEN
POINT
POS
RIGHT$
RND
RTRIM$
DATE$
ENVIRON$
EOF
ERDEV
ERDEV$
ERL
ERR
EXP
FILEATTR
FIX
FRE
FREEFILE
HEX$
INKEY$
INP
INPUT$
SCREEN
SEEK
SGN
SIN
SPACE$
SPC
SQR
STICK
STR$
STRIG
STRING$
INSTR
INT
IOCTL$
LBOUND
LCASE$
LEFT$
LEN
LOC
LOF
LOG
LPOS
LTRIM$
MID$
MKDMBF$
MKD$
MKI$
TAB
TAN
TIME$
UBOUND
UCASE$
VAL
VARPTR
VARPTR$
VARSEG

APPENDICE D

Conversione di programmi da GW-BASIC

In questa appendice vengono descritte le differenze tra il linguaggio di programmazione GW-Basic, fornito con le precedenti versioni di MS-DOS, e QBasic, fornito con la versione 5 di MS-DOS. In particolare cercheremo di dare dei suggerimenti sulla conversione dei programmi in GW-Basic per poter essere eseguiti sotto MS-DOS QBasic. Si può subito notare che QBasic è sostanzialmente compatibile con GW-Basic per cui la maggior parte dei programmi in GW-Basic possono essere eseguiti sotto QBasic senza alcuna modifica. A volte, tuttavia, può essere necessario apportare alcune modifiche affinché i programmi vengano eseguiti correttamente.

VERSIONI PRECEDENTI DI QBASIC

Nell'industria dei personal computer c'è sempre stata la tradizione di distribuire una versione di Basic assieme a MS-DOS; la ragione sta nel fatto che sin dalle origini si era capito che non sempre i prodotti software in commercio avrebbero soddisfatto le esigenze degli utenti e che di tanto in tanto questi avrebbero avuto necessità di scrivere i propri programmi.

I limiti strutturali dei primi PC si riflettevano sulle prime versioni di Basic con essi distribuiti (BASIC, GW-BASIC, BASICA): queste dovevano infatti essere versioni ridotte e compatte del linguaggio. Nel tempo però Basic si è evoluto in un ambiente di programmazione che supporta funzionalità molto avanzate proprie dei linguaggi di programmazione professionali, e capacità di modifica proprie di potenti elaboratori di testo. QBasic contiene quasi tutte le funzionalità del più potente Microsoft QuickBasic Compiler (il più venduto linguaggio di programmazione di Microsoft) ma mantiene la compatibilità con versioni di Basic precedenti.

NOTA: Le versioni di Basic distribuite con i PC nell'arco di diversi anni hanno sempre contenuto delle modifiche e dei miglioramenti. Per semplicità indicheremo tutte le versioni di Basic distribuite con MS-DOS prima della versione GW-BASIC di MS-DOS 5. Per vedere più in dettaglio le differenze tra la vostra versione di Basic e QBasic fate riferimento alla documentazione del vostro linguaggio.

Qual'è la differenza?

MS-DOS QBasic differisce da GW-BASIC per molti aspetti. Le sue avanzate funzionalità, aggiunte alla compatibilità con le versioni precedenti, ne fanno un degno successore di GW-BASIC.

- **QBasic è un linguaggio più ricco.** Contiene infatti un maggior numero di enunciati, di funzioni e di altri elementi del linguaggio, dando all'utente un maggior controllo sul computer e la possibilità di creare programmi più efficienti.
- **QBasic dispone di un migliore ambiente di modifica.** L'ambiente di programmazione basato sui menu contiene molte funzionalità tipiche degli elaboratori o editor di testo più avanzati. Inoltre QBasic supporta comandi di modifica attivabili con la tastiera o con il mouse e per il debug.
- **QBasic ha una documentazione in linea.** Il sistema di guida in linea di QBasic fornisce una documentazione completa sul linguaggio e delle indicazioni sull'uso dell'ambiente di modifica.

QBasic è più veloce di GW-BASIC. L'avanzato "threaded p-code interpreter"" di QBasic permette di eseguire i programmi molto più velocemente che con GW-BASIC. Inoltre gli strumenti di programmazione di QBasic rendono lo stesso processo di sviluppo più veloce (scrittura, modifica e debug).

ESECUZIONE DI PROGRAMMI GW-BASIC IN QBASIC

Per eseguire un programma scritto in GW-BASIC sotto QBasic basta salvare il programma in formato ASCII, aprirlo dall'interno di QBasic ed eseguirlo. La maggior parte dei programmi in GW-BASIC non richiedono alcuna modifica per essere eseguiti in QBasic (noterete però che vengono eseguiti più velocemente in QBasic che in GW-BASIC).

Se QBasic non è in grado di eseguire un programma scritto in GW-BASIC mostra un messaggio di errore di sintassi in una finestra di dialogo in prossimità del primo problema trovato. Se ciò accade sarà necessario modificare il codice del programma per adattarlo alle regole di QBasic.

Salvare un programma GW-BASIC in formato ASCII

Tutti i programmi in QBasic devono essere salvati in formato ASCII (solo testo). Per definizione l'editor di GW-BASIC salva i programmi in uno speciale formato compresso che QBasic non è in grado di leggere; per questa ragione è necessario salvarli in formato ASCII, nel modo seguente:

1. Eseguire GW-BASIC (si ricordi che MS-DOS 5 non dispone di questo programma e perciò è necessario reperirlo in una precedente versione di MS-DOS).

2. Usare il comando LOAD per caricare il programma che si vuole eseguire.

3. Usare il comando LIST per verificare i contenuti del programma.

4. Usare il comando SAVE e l'opzione A per salvare il programma; per esempio, per salvare un programma dal nome test.BAS in formato ASCII basta digitare la seguente riga di comando in GW-BASIC:

```
save "test.bas", a
```

5. Uscire da GW-BASIC usando questa riga di comando:

```
system
```

La prima figura di pagina seguente mostra il processo di carico, visualizzazione, salvataggio e uscita sullo schermo di GW-BASIC

Esecuzione di un programma GW-BASIC in QBasic

Dopo aver salvato il programma in formato ASCII lo si può caricare ed eseguire in QBasic; per caricare il programma TEST.BAS si può digitare questa riga di comando al prompt di MS-DOS:

```
C:\>qbasic test.bas
```

Otterrete un risultato simile a quello mostrato nella seconda figura di pagina seguente.

Se il listato non è chiaro e contiene molti caratteri di controllo e caratteri del set IBM significa che il programma probabilmente non è in formato ASCII. In questo caso si deve uscire da QBasic e salvare il programma in formato ASCII, nel modo descritto in precedenza.


```
GW-BASIC 3.20
(C) Copyright Microsoft 1983,1984,1985,1986
60332 Bytes free
Ok
LOAD"INTEREST.BAS"
Ok
LIST
10 INPUT "PRINCIPAL                         Lit ",PRINCIPAL
20 INPUT "ANNUAL INTEREST RATE(%)?             ",ANNUALRATE
30 INPUT "NUMBER OF MONTHS?                    ",MONTHS
40 MONTHRATE = ANNUALRATE/12
50 FACTOR = (MONTHRATE/100)+1
60 PRINT "THE COMPUNDING FACTOR IS "FACTOR
70 FOR N = 1 TO MONTHS
80 PRINCIPAL = PRINCIPAL * FACTOR
90 PRINT "AFTER MONTH "N" THE NEW PRINCIPAL IS Lit"PRINCIPAL
100 NEXT N
Ok
SAVE "INTEREST.BAS",A
Ok
SYSTEM

1LIST  2RUN←  3LOAD"  4SAVE"  5CONT←  6,"LPT1  7TRON←  8TROFF←  9KEY  0SCREEN
```

```
File  Modifica  Visualizza  Cerca  Esegui  Debug  Opzioni                  ?
                                   INTEREST.BAS
10 INPUT "PRINCIPAL                         Lit ", PRINCIPAL
20 INPUT "ANNUAL INTEREST RATE(%)?             ", ANNUALRATE
30 INPUT "NUMBER OF MONTHS?                    ", MONTHS
40 MONTHRATE = ANNUALRATE / 12
50 FACTOR = (MONTHRATE / 100) + 1
60 PRINT "THE COMPUNDING FACTOR IS "; FACTOR
70 FOR N = 1 TO MONTHS
80 PRINCIPAL = PRINCIPAL * FACTOR
90 PRINT "AFTER MONTH "; N; " THE NEW PRINCIPAL IS Lit"; PRINCIPAL
100 NEXT N

                               Immediato

<Maius+F1=Guida><F6=Finestra><F2=Subs><F5=Esegui><F8=Passo>     M  00001:001
```

Per eseguire il programma TEST.BAS in QBasic basta aprire il menu Esegui e scegliere Avvia; se il programma è pienamente compatibile dovrebbe essere eseguito senza problemi.

Cosa succede se non viene eseguito?

Se il programma in GW-Basic contiene un errore di sintassi, l'esecuzione viene interrotta e appare un messaggio di errore che descrive la natura e il luogo del problema. Correggere un errore di sintassi può essere facile, ma spesso richiede una certa conoscenza del linguaggio e anche buone doti "investigative". Il miglior posto per ottenere informazioni è il sistema di aiuto in linea di QBasic che contiene informazioni dettagliate su ogni tipo di errore di sintassi e una completa documentazione per ogni elemento di QBasic. Per avere maggior informazioni su uno specifico errore di sintassi basta premere F1 al momento in cui esso appare sullo schermo. Una volta compresa la natura del problema basta collocare il cursore sull'enunciato o funzione associata a quell'errore e premere nuovamente F1 per vedere la documentazione relativa. Con tutte queste informazioni si dovrebbe essere in grado di capire come correggere l'errore ed eseguire il programma con successo.

Correzione di errori tipici

Sebbene il processo di conversione di un programma da GW-BASIC in QBasic possa sembrare più un'arte che una scienza ci sono alcuni problemi che possono ricorrere con una certa frequenza; in questa sezione cercheremo di vedere i più importanti.

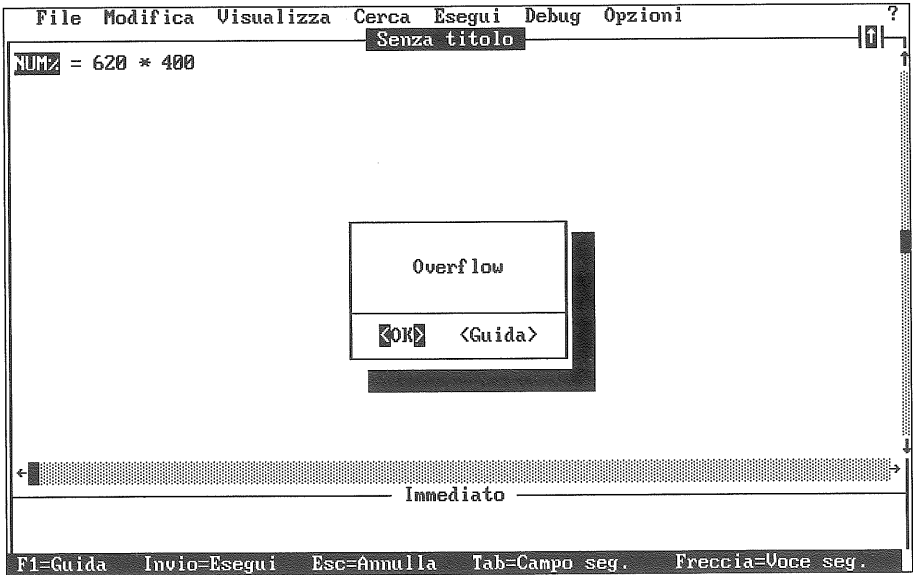
Errori generali di sintassi

A volte l'errore di sintassi che impedisce al programma di essere eseguito è una semplice incompatibilità nella punteggiatura usata tra elementi di un enunciato (uso incompatibile di virgole, punti e virgola o Invio). Altre volte l'errore consiste in lievi differenze. tra la sintassi di GW-BASIC e quella leggermente diversa di QBasic. Per correggere questi errori è molto utile fare riferimento alla documentazione in linea e confrontarla attentamente con la sintassi dell'elemento di GW-BASIC che contiene l'errore in questione. Spesso la differenza di sintassi sarà evidente e sarà altrettanto chiaro quale apportare.

Prima di rischiare

Se il programma che cercate di convertire non è stato scritto da voi (e perciò non sapete esattamente come lavora) può risultare molto più difficile correggere un problema di incompatibilità. Se avete intenzione di fare degli esperimenti accertatevi di avere una copia di riserva del programma; in questo modo potete sempre ricominciare dall'inizio se le cose non vanno per il meglio. Può essere anche utile studiare un listato del programma per capire esattamente come agisce il codice prima di tentare una conversione dettagliata.

Gestione dei tipi di variabile

Si noti che le dimensioni e i tipi di variabili, specialmente gli interi, variano tra le diverse versioni di Basic. Se QBasic trova dei problemi nel modo in cui le variabili vengono usate in un programma, farà apparire un messaggio di errore; questo messaggio appare, per esempio, se una variabile intera di tipo *regular* è troppo piccola per contenere il numero assegnatole:

Se ciò accade basta cambiare il tipo di variabile intera utilizzata. Per esempio, la seguente assegnazione di un valore intero a una variabile intera di tipo regular in un programma corretto in Hewlett-Packard (HP) BASIC crea un errore di "overflow" in QBasic:

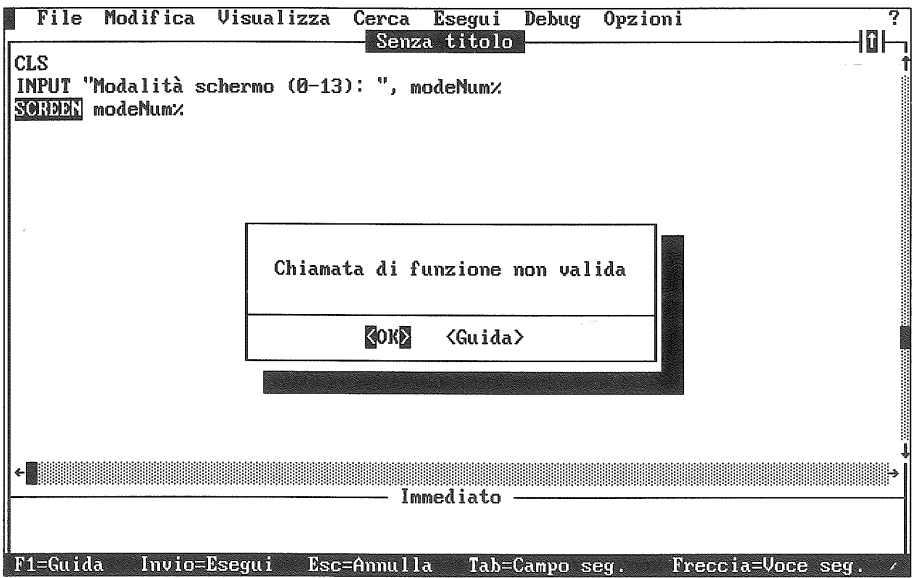
```
NUM% = 620*400
```

Quando convertite un programma in QBasic sarà necessario cambiare il tipo di variabile per garantire uno spazio maggiore:

```
NUM& = 620*400
```

Quando cambiate una variabile ricordate di modificare il tipo adottato in tutto il programma (QBasic tiene separate variabili con lo stesso nome ma di diverso tipo).

Impostazione della modalità schermo corretta

Molti programmi in GW-BASIC, come vecchi giochi per computer, erano progettati per supportare modalità video speciali che non sono supportate da QBasic o dagli adattatori video oggi in commercio. Se eseguite un programma che ha una modalità video non supportata da QBasic o dal vostro adattatore video, otterrete il seguente messaggio di errore:

Per correggere questo problema bisogna fare in modo che il programma utilizzi una modalità supportata da QBasic e compatibile con l'adattatore video del vostro computer (si veda il capitolo 11 per avere maggiori informazioni sugli adattatori video e sull'enunciato SCREEN). Si noti che il programma in GW-BASIC contiene molti comandi grafici, può risultare quindi necessario convertire alcune coordinate dello schermo presenti nel programma per adattarle alla nuova modalità schermo.

Come ottenere i colori desiderati

I colori di sfondo e di primo piano disponibili in GW-BASIC e in QBasic dipendono dagli enunciati PAINT, PALETTE, SCREEN e COLOR; gli intervalli e i significati degli argomenti di questi enunciati sono cambiati con il passare degli anni e perciò alcuni programmi in GW-BASIC potrebbero mostrare colori diversi quando eseguiti in QBasic. Consultate la documentazione on-line di QBasic su PAINT, PALETTE, SCREEN e COLOR per aggiornare il programma con i valori di colore desiderati. Si noti che con l'evoluzione degli schermi avvenuta in questi anni, valori di colore e modalità schermo addizionali potranno essere disponibili al momento della conversione.

Esecuzione troppo veloce

Un problema comune a vecchi programmi in GW-BASIC (specialmente giochi) sta nel fatto che QBasic è in grado di eseguirli più velocemente. La velocità può essere un problema quando un'applicazione è stata disegnata per essere eseguita a una velocità stabilita; ciò è particolarmente vero per programmi scritti in GW-BASIC, quando vengono eseguiti in QBasic, magari con macchine più potenti e veloci.

Se si vuole rallentare un programma scritto per un vecchio computer e con il più lento GW-BASIC basta aggiungere uno o più loop temporali al programma. Un loop temporale è un loop FOR che si ripete molte volte in un programma senza avere alcun effetto tranne quello di far passare il tempo (per cui un loop temporale non fa altro che rallentare l'azione per dare all'utente il tempo di assimilare ciò che è accaduto sullo schermo). Un tipico loop temporale ha un valore di partenza di 1 e un alto valore finale:

```
FOR I% = 1 TO 2000
NEXT I%
```

Durante il loop non accadrà nulla, ma al suo termine sarà trascorso un tempo sufficiente a rallentare, almeno un po', le cose.

Se si sceglie di usare un loop temporale è consigliabile porlo subito dopo la parte di codice che visualizza il risultato sullo schermo; trovare il tempo giusto può richiedere un certo tempo e più di un tentativo a seconda della velocità del computer e dell'effetto che si vuole ottenere dal loop.

TRARRE VANTAGGIO DA QBASIC

Eseguire dei programmi GW-BASIC in QBasic è solo un primo passo nel trarre vantaggio dall'uso di QBasic. In realtà QBasic dispone anche di molte funzionalità nuove e potenti che lo rendono non solo più semplice e veloce da imparare ma anche in grado di rendere i vostri programmi più facili da capire e da correggere in futuro. Le funzionalità elencate di seguito (non disponibili in GW-BASIC) sono state descritte nel corso dei precedenti capitoli e sono certamente in grado di rendere i vostri programmi più evoluti.

- **Tipi di variabili e costanti estese.** QBasic supporta nomi di variabili lunghi 40 caratteri in qualsiasi combinazione di maiuscole e minuscole. QBasic supporta anche variabili intere di tipo *long*, variabili stringa a lunghezza fissa oltre a costanti numeriche e stringa (per maggior informazioni su questi tipi di variabili e costanti si veda il capitolo 4).

- **Nuove strutture decisionali.** Gli enunciati IF multirighe sono una novità di QBasic e possono contenere un numero illimitato di clausole ELSEIF per aiutarvi a controllare il percorso delle funzioni del programma. L'enunciato SELECT CASE rappresenta un nuovo tipo di ramificazione condizionale basato sul valore di una variabile (per maggiori informazioni su queste affermazioni si veda il capitolo 5).

- **Loop migliori.** Oltre al loop FOR QBasic supporta anche il loop DO, per l'esecuzione ripetuta di un blocco di codice fino al verificarsi di una certa condizione (per maggiori informazioni si veda il capitolo 6).

- **Sottoprogrammi e funzioni definite dall'utente.** QBasic supporta blocchi di codice multiriga (con variabili locali e indipendenti), detti sottoprogrammi e funzioni, che possono essere eseguiti molte volte senza dover essere riscritti. Questi sottoprogrammi e funzioni sostituiscono le routine GOSUB e le funzioni a riga singola DEF FN tipiche dei programmi in GW-BASIC (per maggiori informazioni si veda il capitolo 7).

- **Miglioramenti dello stile e dell'ambiente.** QBasic include un sistema interno di controllo della sintassi che avverte l'utente non appena questi commette un simile errore, aiutandolo così a organizzare i programmi e a renderli più semplici da leggere (per esempio evidenziando automaticamente in maiuscolo le parole chiave, rientrando i loop e garantendo coerenza nei nomi delle variabili). QBasic non richiede la numerazione di riga (come GW-BASIC); perciò si possono creare programmi più facili da leggere semplicemente in base a un'attenta pianificazione delle strutture, dei loop e delle procedure, senza bisogno di GOTO e GOSUB tipici di GW-BASIC (si veda il capitolo 2).

APPENDICE E

Soluzioni alle domande e agli esercizi

Questa appendice contiene le risposte alle domande e agli esercizi dei capitoli dal 2 al 12.

CAPITOLO 2

1. Digitare *qbasic* al prompt di sistema.
2. Falso.
3. La Finestra di digitazione contiene il programma ed è la finestra principale di QBasic. La Finestra Immediato permette di controllare le righe di programma prima di utilizzarle.
4. Il comando Nuovo pulisce la Finestra di digitazione e prepara QBasic per un nuovo programma. Il comando Apri apre la relativa finestra di dialogo per caricare da disco un programma esistente.
5. La Finestra di digitazione mostra il programma su cui si sta lavorando. Lo schermo di output mostra il risultato del programma al momento dell'esecuzione.
6. Il comando Taglia cancella un blocco di testo selezionato dalla Finestra di digitazione e lo pone negli Appunti. Il comando Copia pone una copia del blocco di testo selezionato negli Appunti senza rimuovere l'originale dalla Finestra di digitazione.
7. Si può cancellare testo da un programma nei modi seguenti:
   - Col comando Taglia dal menu Visualizza (testo selezionato)
   - Col tasto Canc (un singolo carattere o testo selezionato)
   - Col tasto Backspace (un singolo carattere)
   - Con la combinazione di tasti Ctrl-Y (una riga)
8. Il cursore di inserimento aggiunge testo senza sovrascrivere i caratteri esistenti. Il cursore di sovrascrittura sostituisce i caratteri esistenti con quelli digitati. La differenza a video tra i due è che il cursore di inserimento è rappresentato da un simbolo di sottolineatura lampeggiante mentre il cursore di sovrascrittura appare come un rettangolo lampeggiante. Per alternare l'uso dei due cursori si usa il tasto Ins.
9. Otto.

10. I seguenti nomi di file non sono validi:

Nome di file non valido	Ragione
NOMESTAMPA.BAS	Troppo lungo (NOMESTAMPA.BAS è composto di 10 caratteri)
BLU+BLU.BAS	Il carattere + non è ammesso
BLU BLU. BAS	Lo spazio non è ammesso
ABC-E-DEF.BAS	Troppo lungo (ABC-E-DEF.BAS è composto di 9 caratteri)
PROG[1].BAS	I caratteri *[* e *]* non sono ammessi
1*PROG.BAS	Il carattere jolly * non è ammesso

11. Selezionare Esci dal menu File per uscire da QBasic. Se il file è stato modificato QBasic mostra una finestra di dialogo che contiene le seguenti opzioni: *Sì* (salva il programma), *No* (scarta i cambiamenti non salvati), *Annulla* (ritorna in QBasic) e *Guida* (mostra la Guida on-line).

CAPITOLO 3

1. Un enunciato opera in modo diretto, producendo generalmente risultati ovvi o tangibili (per esempio mostra un carattere sullo schermo). Una funzione generalmente svolge il suo lavoro dietro le quinte e restituisce un valore al programma che può essere usato come argomento per un enunciato.
2. BEEP — enunciato
   CLS — enunciato
   DATE$ — funzione
   PRINT — enunciato
   TIME$ — funzione
3. Un elemento tra parentesi quadre è facoltativo. Il carattere | indica che si può scegliere solo uno dei valori tra parentesi.
4. Un argomento è una parte di informazione fornita a un enunciato o a una funzione.
5. Una stringa è una raccolta di caratteri (che può includere lettere, numeri e simboli) racchiusa tra virgolette. Un'espressione numerica è un numero o

una variabile numerica (o, come visto nel capitolo 4, una qualsiasi espressione che produce un risultato numerico).

6. Il risultato dei due enunciati PRINT è lo stesso.

7. Un punto e virgola o una virgola alla fine dell'enunciato PRINT fa sì che il risultato del successivo enunciato PRINT venga mostrato sulla stessa riga. Un punto e virgola fa sì che questo risultato appaia immediatamente dopo quello dell'enunciato PRINT corrente; una virgola fa sì che questo risultato appaia nella successiva area di stampa.

CAPITOLO 4

1. Intero di tipo *regular*, intero *long*, virgola mobile a precisione singola e virgola mobile a precisione doppia. Differiscono nel tipo e nella dimensione del numero che possono contenere e nei rispettivi di dichiarazione del tipo.

2. Per riservare uno spazio per un segno meno nel caso in cui il numero sia stato o possa diventare negativo.

3. È probabile che una virgola non ammessa sia apparsa all'interno di un numero.

4. È stato assegnato alla variabile un numero al di fuori dell'intervallo ammesso per quel tipo di dati numerici.

5. a. Intero regolare
   b. Virgola mobile a precisione singola
   c. Virgola mobile a precisione singola
   d. Virgola mobile a precisione doppia
   e. Intero *regular* o virgola mobile a precisione singola
   f. Virgola mobile a precisione singola
   g. Intero *long*
   h. Virgola mobile a precisione doppia
   i. Virgola mobile a precisione singola

6. La normale divisione restituisce un risultato frazionale; la divisione intera restituisce solo un intero, scartando il resto; la divisione con resto restituisce solo il resto.

7. Esponenziale (^); moltiplicazione e divisione (*, /, \ , MOD); addizione e sottrazione (+, -).

8. 300.

9. Una possibile soluzione a questo problema è il programma CALC.BAS:

```
' CALC.BAS
' Questo programma calcola e stampa quattro formule

CLS

PRINT "ABS(-10) + 5 ="; ABS(-10) + 5
PRINT "SQR(36) ="; SQR(36)
PRINT "SQR(4) n^ 2 ="; SQR(4) ^ 2
PRINT "COS(3.14) ="; COS(3.141592654#)
```

10. Una possibile soluzione a questo problema è il programma CERCHIO.BAS:

```
' CERCHIO.BAS
' Questo programma mostra la circonferenza di un cerchio
' quando il raggio è fornito dall'utente
CONST PI# = 3.141592654#     ' dichiara la costante

CLS   ' Pulisce lo schermo

PRINT "Questo programma calcola la circonferenza di un cerchio
";
PRINT "dal suo raggio."
PRINT ' riceve l'input dall'utente

INPUT "Inserire il raggio del cerchio: ", raggio!

circon! = 2 * PI# * raggio!  ' calcola la circonferenza
PRINT ' stampa il risultato
PRINT "La circonferenza del cerchio è"; circon!
```

CAPITOLO 5

1. Un'espressione numerica usa operatori matematici e fornisce un risultato numerico. Un'espressione condizionale usa operatori condizionali e fornisce un risultato del tipo Vero o Falso.
2. b, c, f, i, j, k.
3. Vero.
4. AND richiede che entrambe le condizioni siano vere affinché un'azione venga intrapresa; OR richiede che solo una condizione sia vera affinché un'azione venga intrapresa.
5. ELSE permette di eseguire un blocco di enunciati quando un'espressione condizionale in un enunciato IF viene valutata come falsa. ELSE è l'opposto di THEN.
6. ELSEIF permette di valutare un'altra condizione dopo che una precedente affermazione IF o ELSEIF è stata valutata come falsa. La parola chiave THEN deve apparire sulla stessa riga di ELSEIF.
7. Una possibile soluzione a questo problema è il programma DOMANDA.BAS:

```
' DOMANDA.BAS
' Questo programma pone una domanda sulla programmazione
' in QBasic.

CLS

INPUT "Ti piace programmare in QBasic (S/N)? ", risposta$
PRINT

IF (risposta$ = "S") OR (risposta$ = "s") THEN
     PRINT "Bene! C'è ancora tanto da imparare!"
ELSEIF (risposta$ = "N") OR (risposta$ = "n") THEN
     PRINT "Mi spiace. Ma non ti scoraggiare."
ELSE
     PRINT "Eseguite il programma di nuovo."
     PRINT "Scrivete 'S' per Si e 'N' per No."
END IF
```


8. QBasic valuta l'espressione che appare dopo SELECT CASE e controlla la clausola CASE che segue per trovare un valore uguale. QBasic esegue le affermazioni che seguono la clausola CASE i cui valori sono uguali all'espressione nell'affermazione SELECT CASE.
9. CASE ELSE permette di specificare un blocco di enunciati che verranno eseguiti se tutte le condizioni CASE in un enunciato SELECT CASE sono valutate come false.
10. IF permette di usare un'espressione condizionale in un enunciato CASE. TO permette di specificare un intervallo di valori numerici in un enunciato CASE.
11. Una possibile soluzione a questo problema è il programma STATI.BAS:

```
' STATI.BAS
' Questo programma mostra informazioni su tre diversi stati
degli USA

CLS

PRINT "Su quale dei seguenti stati volete delle informazioni?"
PRINT
PRINT "  1)Washington"
PRINT "  2)Virginia"
PRINT "  3)Minnesota"
PRINT
INPUT "Stato (1-3): ", risposta%
PRINT

SELECT CASE risposta%
     CASE 1
          PRINT "** Washington **"
          PRINT "  Popolazione nel 1980: 4538000"
          PRINT "  Capitale: Olympia"
          PRINT "  Anno di fondazione: 1889"
     CASE 2
          PRINT "** Virginia **"
          PRINT "  Popolazione nel 1980: 5346818"
          PRINT "  Capitale: Richmond"
          PRINT "  Anno di fondazione: 1788"
```

```
    CASE 3
         PRINT "** Minnesota **"
         PRINT "  Popolazione nel 1980: 4075970"
         PRINT "  Capitale: St. Paul"
         PRINT "  Anno di fondazione: 1858"
    CASE ELSE
         PRINT "Eseguite il programma di nuovo e selezionate ";
         PRINT "un numero da 1 a 3"
END SELECT
```

CAPITOLO 6

1. La variabile di calcolo identifica il valore del loop (un intero tra i limiti di inizio e fine del loop).

2. *inizio* e *fine* possono essere costanti , variabili o espressioni numeriche. I valori possono essere negativi, positivi o uguali a zero.

3. 4.

4. L'enunciato SOUND fa sì che lo speaker del computer emetta una tonalità di una certa frequenza e durata.

5. Un loop nidificato è un loop di tipo FOR, WHILE o DO all'interno di un altro loop di tipo FOR, WHILE o DO.

6. Posizionare la condizione all'inizio del loop DO assicura che essa debba essere verificata prima che il loop venga eseguito. Posizionare la condizione alla fine assicura che le affermazioni nel loop vengano eseguite almeno una volta.

7. Usare un loop di tipo FOR quando si vuole eseguire un blocco di enunciati uno specifico numero di volte. Usare un loop di tipo WHILE o DO per eseguire un blocco di affermazioni basato sul valore di una condizione.

8. Una possibile soluzione a questo problema è il programma BENZINA.BAS.


```
' BENZINA.BAS
' Questo programma usa un loop di tipo FOR per registrare le
' spese settimanali in benzina.

CLS

PRINT "Inserire l'ammontare speso in benzina ";
PRINT "per ogni giorno della settimana."
PRINT

FOR giorno% = 1 TO 7
     PRINT "  Soldi spesi nel giorno"; giorno%
     INPUT "--> L", giornoTotale!
     settimTotale! = settimTotale! + giornoTotale!
NEXT giorno%

PRINT
PRINT "Hai speso L"; settimTotale! "in benzina questa
settimana!"
```

9. Una possibile soluzione a questo problema è il programma SUONO.BAS

```
' SUONO.BAS
' Questo programma suona una nota in base ai valori di frequenza
' e durata definiti dall'utente.
CLS

PRINT "Inserire i valori di frequenza e durata per il suono che
si vuole"
PRINT "ascoltare. Per uscire scrivere -999 come frequenza."
PRINT

DO
     INPUT "Frequenza (37-32767): ", frequenza%
     IF (frequenza% <> -999) THEN
          INPUT "Durata(0-65535): ", durata&
          SOUND frequenza%, durata%
          PRINT
     END IF
LOOP UNTIL (frequenza% = -999)
```

10. Una possibile soluzione a questo problema è il programma DADI.BAS.

```
' DADI.BAS
' Questo programma tira un dado 10 volte e mostra il messaggio
' "Bel tiro!" se esce il numero 6.

CLS

INPUT "Premere Invio per tirare il dado 10 volte.", richiesta$
PRINT

RANDOMIZE TIMER

FOR i% = 1 TO 10
     tiro% = INT(RND * 6) + 1
     PRINT "Tiro: "; tiro%,
     IF (tiro% = 6) THEN
          COLOR 2
          PRINT "Bel tiro!"
          COLOR 7
     ELSE
          PRINT
     END IF
NEXT i%
```

CAPITOLO 7

1. Sono tutti dei vantaggi.

2. Il nome del sottoprogramma *ScriviNome$* contiene un carattere tipo dichiarazione stringa ($). Questo è un errore (solo le funzioni vanno marcate col tipo di dato che restituiranno). Una corretta affermazione SUB sarebbe la seguente:

   ```
   SUB ScriviNome (Nome$, Cognome$)
   ```

3. F2.

4. Il comando Procedura passo nel menu Debug (F10).

5. `COMMON SHARED totale%`

6. Un sottoprogramma permette di eseguire vari compiti o di inviare un numero di valori al programma principale. Una funzione permette di eseguire un compito più piccolo e specifico e restituisce un solo valore al programma principale.

7. Il sottoprogramma *InfoAuto* potrebbe assomigliare al seguente:

   ```
   SUB InfoAuto (marca$, modello$, anno%, colore$)

   PRINT "Scrivete le informazioni sulla vostra macchina"
   PRINT
   INPUT "  Produttore: ", marca$
   INPUT "  Modello: ", modello$
   INPUT "  Anno: ", anno%
   INPUT "  Colore: ", colore$

   END SUB
   ```

   Un'affermazione che chiama il sottoprogramma *InfoAuto* potrebbe essere questa:

   ```
   InfoAuto marca$, modello$, anno%, colore$
   ```

8. La funzione *TrovaMaggiore%* può essere scritta in questo modo:

   ```
   FUNCTION TrovaMaggiore% (int1%, int2%)

   IF (int1% >= int2%) THEN
     TrovaMaggiore% = int1%
   ELSE
     TrovaMaggiore% = int2%
   END IF

   END FUNCTION
   ```

   Un enunciato che chiama la funzione *TrovaMaggiore%* potrebbe essere questa:

   ```
   PRINT "L'intero più grande è"; TrovaMaggiore%(primo%, secondo%)
   ```

9. Una possibile soluzione a questo problema è il programma FORME.BAS.

```
' FORME.BAS
' Questo programma mostra una sagoma riempita con un
' carattere scelto.
' Dichiara le costanti usate come argomenti per
' l'affermazione COLOR

CONST CYAN% = 3
CONST WHITE% = 7

' dichiara i sottoprogrammi prima che vengano usati;
' i nomi e i parametri
' del sottoprogramma dovrebbero essere uguali a quelli
' nei sottoprogrammi

DECLARE SUB Forma (simbolo$, scelta%)
DECLARE SUB StampaRiga (car$)
DECLARE SUB StampaRettangolo (car$)
DECLARE SUB StampaTriangolo (car$)
CLS

' chiama il sottoprogramma Forma per avere il carattere
' e la forma desiderati

Forma carattere$, forma%   ' trasferisce i due argomenti a Forma

PRINT
PRINT
COLOR CYAN%
' usa l'affermazione CASE per chiamare il sottoprogramma
' richiesto

SELECT CASE forma%
      CASE 1      ' se forma% = 1 mostra un triangolo
            StampaTriangolo carattere$
      CASE 2      ' se forma% = 2 mostra un rettangolo
            StampaRettangolo carattere$
      CASE 3      ' se forma% = 3 mostra una riga
            StampaRiga carattere$
END SELECT

COLOR WHITE%
```



```
END

SUB Forma (simbolo$, scelta%)
' Il sottoprogramma Forma chiede all'utente un simbolo
' e una forma e li restituisce al programma principale
' nelle variabili simbolo$ e scelta%

PRINT " Questo programma stampa una raccolta di caratteri nella ";
PRINT " forma specificata."
PRINT
INPUT "Che carattere vorresti usare? ", simbolo$
PRINT
PRINT "Che forma vorresti vedere?"
PRINT
PRINT "  1) Triangolo"
PRINT "  2) Rettangolo"
PRINT "  3) Riga"
PRINT

DO     ' ripete la richiesta fino a che scelta% è nel
       ' giusto intervallo
      INPUT "Forma (1, 2 o 3): ", scelta%

LOOP WHILE (scelta% < 1) OR (scelta% > 3)

END SUB     ' il sottoprogramma è completo e ritorna al programma
            ' principale

SUB StampaRiga (car$)

' Questo programma riceve un argomento dal programma principale
' e lo usa per stampare una riga lunga 30 caratteri.

CONST LENGTH% = 30     ' imposta la lunghezza della riga
                       ' in 30 caratteri

FOR i% + 1 TO LENGTH%  ' mostra il carattere 30 volte
      PRINT car$;      ' usa il punto e virgola per stampare
                       ' i caratteri uno dopo l'altro
NEXT i%
```

```
PRINT

END SUB

SUB StampaRettangolo (car$)

' Questo programma riceve un argomento dal programma principale
' e lo usa per stampare un rettangolo lungo 50 caratteri
' e alto 7.

CONST LENGTH% = 50      ' imposta la lunghezza della rettangolo
                        ' a 50
CONST HEIGHT% = 7       ' imposta l'altezza del rettangolo a 7

FOR i% = 1 TO HEIGHT%   ' per ognuna delle sette righe nel
                        ' rettangolo,
      FOR j% + 1 TO LENGTH%  ' mostra 50 caratteri uno dopo
                             ' l'altro
            PRINT car$;
      NEXT j%
      PRINT j%     ' va a capo dopo ogni riga
NEXT i%
END SUB

SUB StampaTriangolo (car$)

' Questo programma riceve un argomento dal programma principale
' e lo usa per stampare un triangolo equilatero. La funzione Tab
' sposta il cursore nella corretta posizione di colonna.

CONST ROWS% = 10 ' imposta il numero di righe a 10
sinistra% = ROWS%       ' usa sinistra% per costruire il lato
                        ' sinistro del triangolo
destra% = ROWS% + 1     ' usa destra% per costruire il lato destro
                        ' del triangolo

FOR calcoloRiga% = 1 TO ROWS%     ' per ogni riga nel triangolo
      FOR i% = sinistra% TO ROWS%
            PRINT TAB(i%); car$;  ' mostra il lato sinistro
                                  ' della riga
NEXT i%
```



```
FOR i% = ROWS% + 1 TO destra% - 1        ' mostra il lato destro
                                         ' della riga
      PRINT TAB(i%); car$;
      NEXT i%
      PRINT ' va a capo dopo ogni riga
      sinistra% = sinistra - 1      ' il primo carattere nella riga
                                    ' successiva
      destra% = destra% -      ' creerà uno spazio vicino
                               ' al margine sinistro
NEXT calcoloRiga%' e uno vicino a quello destro

END SUB
```

CAPITOLO 8

1. Gli enunciati READ per convenzione.
2. DATA, READ e RESTORE sono più indicati per dati che ricadono in queste categorie:
   - Dati conosciuti in anticipo (prima dell'esecuzione del programma).
   - Dati che appaiono sempre nello stesso ordine.
   - Dati che possono circolare ripetutamente, come i giorni della settimana.
3. Una possibile soluzione a questo problema è il programma GIGI.BAS:

```
' GIGI.BAS
' Questo programma legge il nome di Gigi Rossi e dei
' suoi amici da un enunciato DATA.
CLS

FOR i% = 1 TO 7
      READ amico$
      PRINT amico$
NEXT i%

DATA Gigi, Memi, Andrea, Carlo, Maria, Lori, Toni
```

4. Falso.
5. *DIM numeri!(99)*
6. Un indicatore di fine dati è un numero o una stringa che indicano che non vi sono ulteriori informazioni in una lista.
7. Una possibile soluzione a questo problema è il programma BANZAI.BAS:

```
' BANZAI.BAS
' Questo programma usa una matrice stringa dinamica
' monodimensionale per conservare
' i nomi dei personaggi del file "Le avventure di Banzai".

OPTION BASE 1    ' imposta la base della matrice a 1
CLS

PRINT "** Questo programma raccoglie i personaggi del film";
PRINT "'Le avventure di Banzai' **"
PRINT
INPUT "Quanti nomi volete inserire? ", nomi%

DIM personaggi$(nomi%) ' dimensiona la matrice

PRINT ' riempie la matrice dinamica
FOR i% = 1 TO nomi%
     INPUT " Nome: ", personaggi$(i%)
NEXT i%

PRINT
PRINT "Avete inserito i nomi seguenti:"
PRINT

FOR i% = 1 TO nomi%    ' stampa i contenuti della matrice
     PRINT personaggi$(i%)
NEXT i%
```

8. Un errore *Indice inferiore fuori dal limite* è un tentativo del programma di far riferimento a un elemento non esistente in una matrice.

CAPITOLO 9

1. Vero.
2. Falso. La dichiarazione corretta non ha il carattere di dichiarazione del tipo *$*:

   `DIM cognome AS STRING * 20`
3. `UNODUETRE`
4. a, b, c, d.
5. Il valore 0 restituito da INSTR può significare:
   - *cercastringa* non è stato trovato in *stringabase*
   - *inizio* è maggiore della lunghezza di *stringabase*
   - *stringabase* non contiene alcun carattere
6. `H`
7. 77.
8. Una possibile soluzione a questo problema è il programma NOMI.BAS:

```
' NOMI.BAS
' Questo programma riceve nome e cognome dall'utente per
' poi mostrarli in maiuscolo.

CLS

INPUT "Nome: ", nome$
INPUT "Cognome: ", cognome$
PRINT
PRINT UCASE(cognome$); ", "; UCASE$(nome$)
```

9. Una possibile soluzione a questo problema è il programma INVERSO.BAS:

```
' INVERSO.BAS
' Questo programma inverte l'ordine dei caratteri in una stringa

CLS

        ' riceve la stringa dall'utente
INPUT " Scrivete la stringa di caratteri da invertire: ", inStringa$
numdiCar% = LEN(inStringa$)     ' individua la lunghezza della stringa

FOR i% = numdiCar% TO 1 STEP -1 ' percorre la stringa a ritroso
        tempCar$ = MID$(inStringa$, i%, 1)    ' estrae una lettera
                                              ' alla volta
        inverso$ = inverso$ + tempCar$  ' costruisce la nuova stringa
NEXT i%

PRINT ' mostra la nuova stringa
PRINT "I caratteri in ordine inverso sono "; inverso$
```

10. Una possibile soluzione a questo problema è il programma DIVIDI.BAS.

```
' DIVIDI.BAS
' Questo programma divide una stringa in tre parti
CONST BLANK$ = " " ' dichiara la costante stringa
car$ = " "   ' inizializza le variabili
calcoloCar% = 1
calcoloNome% = 0
CLS' pulisce lo schermo
PRINT "Inserire il nome in questo formato: primo secondo cognome"
INPUT "Nome: ", nomeIntero$      ' raccoglie le tre parti del
                                 ' nome dall'utente
nomeLunghez% = LEN(nomeIntero$)    ' determina la lunghezza del nome

' Entra in un loop fino a che la stringa in tre parti è stata esaminata

DO WHILE (calcoloCar% <> nomeLunghez% + 1)
' legge i caratteri uno alla volta fino a incontrare uno spazio vuoto
' o la fine di una stringa; assegna i caratteri alla variabile nome$

    DO WHILE (car$ <> BLANK$) AND (calcoloCar% <> (nomeLunghez% + 1))
        car$ = MID$(nomeIntero$, calcoloCar%, 1)
        nome$ = nome$ + car$
        calcoloCar% = calcoloCar% + 1   ' registra il num. del car. letto
    LOOP
```



```
    car$ = "" ' reimposta car$
    calcoloNome% = calcoloNome% + 1 ' incrementa calcoloNome%

    SELECT CASE calcoloNome% ' assegna la stringa alle variabili di nome
        CASE 1 ' in base al valore di calcoloNome%
            primoNome$ = nome$
        CASE 2
            secondoNome$ = nome$
        CASE 3
            cognome$ = nome$
    END SELECT
    nome$ = ""   ' reimposta nome$
LOOP
PRINT
PRINT "Risultati del processo di separazione:"
PRINT
PRINT "Il nome è "; primoNome$  ' mostra i risultati
PRINT "Il secondo nome " "; secondoNome$
PRINT "Il cognome è "; cognome$
```

CAPITOLO 10

1. OUTPUT cancella i contenuti del file esistente; APPEND li aggiunge alla fine del file esistente.
2. d.
3. Falso.
4. LINE INPUT# è più potente di INPUT# quando è necessario leggere da un file lunghe righe di dati stringa o righe contenente virgole.
5. Un argomento letterale dell'enunciato SHELL dev'essere racchiuso tra virgolette. L'enunciato corretto è questo:

   `SHELL "copy test.txt test2.txt"`
6. `Stringa trovata!`
7. Una possibile soluzione a questo problema è il programma PAESE.BAS

```
' PAESE.BAS
' Questo programma conserva un elenco di città in un file sequenziale.

OPEN "PAESE.TXT" FOR OUTPUT AS #1  ' apre il file nella directory/unità
                                   ' corrente

CLS

PRINT "Questo programma conserva nomi di paesi nel file PAESE.TXT."
PRINT "Scrivete i vostri paesi preferiti e digitate FINE per uscire."
PRINT
DO WHILE (paese$ <> "FINE")  ' fino a che l'utente digita FINE
   INPUT "  Nome paese: ", paese$  ' riceve i nomi dall'utente
   IF (paese$ <> "FINE") THEN PRINT #1, paese$  ' trasferisce i nomi
                                              ' in un file
LOOP

CLOSE #1  ' chiude il file

PRINT
INPUT "Premete Invio per vedere le città  inserite. ", richiesta$
PRINT

OPEN "PAESE.TXT" FOR INPUT AS #1   ' apre il file come input
DO WHILE (NOT EOF(1))  ' fino al raggiungimento della fine del file
   INPUT #1, paese$    ' riceve il nome
   PRINT paese$        ' Stampa i nomi
LOOP

CLOSE #1  ' chiude il file
```

8. Una possibile soluzione a questo problema è il programma ORDINA.BAS:

```
' ORDINA.BAS
' Questo programma riceve nomi e indirizzi dall'utente, li
' ordina alfabeticamente in base al cognome e li conserva
' nel file sequenziale NOMI.TXT
DECLARE SUB AggNomiAlFile () ' dichiara le procedure
DECLARE FUNCTION NumeroDiNomiNelFile% ()
DECLARE SUB CopiaFileNelleMatrici (nomi$(), indirizzi$(), numDiOgg%)
DECLARE SUB ShellSort (nomi$(), indirizzi$(), numDiElementi%)
DECLARE SUB CopiaMatriciNelFile (nomi$(), indirizzi$(), numDiOgg%)
DECLARE SUB MostraNuovoFile ()
```



```
OPTION BASE 1   ' imposta la base della matrice a 1

CLS' pulisce lo schermo

AggNomiAlFile   ' chiama il sottoprogramma per ricevere l'input
                ' dall'utente
numDiNomi% = NumeroDiNomiNelFile%     ' chiama la funzione per ottenere
                                      ' il numero di nomi

DIM nomi$(numDiNomi%) ' dimensiona la matrice per contenere i nomi
DIM indirizzi$(numDiNomi%)  ' dimensiona la matrice per contenere
                            ' gli indirizzi

' copia i nomi e gli indirizzi dal file alle matrici in preparazione
' per l'ordinamento
CopiaFileNelleMatrici nomi$(), indirizzi$(), numDiNomi%

' ordina le matrici nomi$ e indirizzi$ alfabeticamente in base
' al cognome
ShellSort nomi$(), indirizzi$(), numDiNomi%

' copia nel file le matrici ordinate
CopiaMatriciNelFile nomi$(), indirizzi$(), numDiNomi%

' mostra il nuovo file sullo schermo
MostraNuovoFile

END

SUB AggNomiAlFile

OPEN "NOMI.TXT" FOR APPEND AS #1   ' apre in modalità append per non
                                   ' perdere nomi e indirizzi esistenti


PRINT "Questo programma aggiunge i nomi e gli indirizzi al file"
PRINT " NOMI.TXT e poi ordina il file alfabeticamente."
PRINT
PRINT "Scrivere i nomi in formato Nome e Cognome. Digitare FINE per
uscire."
PRINT


DO WHILE (nomeIntero$ <> "FINE")   ' riceve nomi fino a quando
                                   ' l'utente digita FINE
```

```
    LINE INPUT " Nome (Cognome, Nome): "; nomeIntero$
    IF (nomeIntero$ <> "FINE") THEN    ' usa LINE INPUT per permettere
                                       ' l'uso di virgole
       PRINT #1, nomeIntero$ ' trasferisce i dati nel file
       LINE INPUT " Indirizzo: "; indirizzo$
       PRINT #1, indirizzo$
    END IF
    PRINT
LOOP

CLOSE #1

END SUB

SUB CopiaMatriciNelFile (nomi$(), indirizzi$(), numDiOgg%)

OPEN "NOMI.TXT" FOR OUTPUT AS #1   ' apre il file come output
                                   ' per sovrascrivere
   ' le voci fuori dall'intervallo

FOR i% = 1 TO numDiOgg%
   PRINT #1, nomi$(i%)    ' trascrive nel file i contenuti della matrice
   PRINT #1, indirizzi$(i%)
NEXT i%

CLOSE #1

END SUB
SUB CopiaFileNelleMatrici (nomi$(), indirizzi$(), numDiOgg%)

OPEN "NOMI.TXT" FOR INPUT AS #1    ' apre il file come input per
                                   ' ricevere i dati

FOR i% = 1 TO numDiOgg%
   LINE INPUT #1, nomi$(i%) ' legge i contenuti del file
   LINE INPUT #1, indirizzi$(i%)
NEXT i%
CLOSE #1
END SUB
SUB MostraNuovoFile
INPUT "Premete Invio per vedere NOMI.TXT.", richiesta$
PRINT


OPEN "NOMI.TXT" FOR INPUT AS #1    ' apre il file come input per
                                   ' ricevere i dati
```



```
DO WHILE (NOT EOF(1)) ' fino a quando raggiunge la fine del file
   LINE INPUT #1, nomeIntero$  ' legge i dati dal file
   LINE INPUT #1, indirizzo$
   PRINT nomeIntero$:  " -- "; indirizzo$   ' e li mostra sullo schermo
LOOP

CLOSE #1

END SUB

FUNCTION NumeroDiNomiNelFile%

OPEN "NOMI.TXT" FOR INPUT AS #1     ' apre il file come input per leggere
                                    ' i dati
calcolo% = 0 ' Registra il numero di elementi nome-e-indirizzo
DO WHILE (NOT EOF(1)) ' fino a quando raggiunge la fine del file
   LINE INPUT #1, nomeIntero$   ' legge il nome e l'indirizzo
   LINE INPUT #1, indirizzo$
   calcolo% = calcolo% + 1   ' aumenta il calcolo degli elementi
LOOP
CLOSE #1
NumeroDiNomiNelFile% = calcolo%     ' restituisce il numero di elementi
                                    ' al programma principale
END FUNCTION
SUB ShellSort (nomi$(), indirizzi$(), numDiElementi%)
' per una descrizione di Shell Sort fate riferimento al capitolo 9.
' si noti che questa versione ordina le due matrici in base
' ai contenuti di nomi$().
span% = numDiElementi% \ 2
DO WHILE (span% > 0)
   FOR i% span% TO numDiElementi% - 1
      j% = i% Ī span% - 1
      FOR j% = (i% Ī span% - 1) TO 1 STEP -span%
      IF nomi%(j%) <= nomi$(j% + span%) THEN EXIT FOR
   ' inverte gli elementi della matrice non ordinati
         SWAP nomi$(j%), nomi$(j% + span%)
         SWAP indirizzi$(j%), indirizzi$(j% + span%)
      NEXT j%
   NEXT i%
   span% = span% \ 2
LOOP
END SUB
```

CAPITOLO 11

1. Le due componenti sono l'adattatore video e lo schermo. L'adattatore video determina la modalità schermo, genera il testo e la grafica da visualizzare oltre ai colori. Lo schermo mostra semplicemente ciò che gli viene inviato dall'adattatore.
2. In modalità testo l'adattatore può visualizzare solo caratteri alfanumerici; in modalità grafica può visualizzare sia testo che forme grafiche (pixel, linee, riquadri, cerchi etc.).
3. LOCATE permette di posizionare il cursore di testo sullo schermo di output.
4. Una possibile soluzione a questo problema è il programma MUOVINOM.BAS.

```
' MUOVINOM.BAS
' Questo programma "sposta" un nome sullo schermo.

CONST DELAY% = 400

CLS

INPUT "Scrivete il vostro nome: ", nome$
LOCATE 10, 1
PRINT nome$

FOR i% = 2 TO 70
      LOCATE 10, i% - 1
      PRINT SPACE$(10)
      LOCATE 10, i%
      PRINT nome$

      FOR j% = 1 TO DELAY%   ' ritarda il loop
      NEXT j%

NEXT i%
```


5. SCREEN cambia la modalità dell'adattatore video e determina la risoluzione dello schermo e (se possibile) il numero di colori che l'adattatore video può visualizzare.
6. Gli enunciati PSET e PRESET impostano singoli pixel sullo schermo nelle posizioni specificate. PSET imposta i pixel nel colore di primo piano corrente; PRESET imposta i pixel nel colore corrente di sfondo.
7. Le coordinate assolute sono calcolate usando come punto di inizio (0, 0), che corrisponde all'angolo in alto a sinistra dello schermo. Le coordinate relative usano l'ultimo punto definito come punto di inizio per il loro calcolo.
8. Vero, solo se si usano le opzioni B e F.
9. Falso.
10. Una possibile soluzione a questo problema è il programma MUOVICER.BAS.

```
' MUOVICER.BAS
' Questo programma "sposta" un cerchio sullo schermo.

CONST DELAY% = 50

SCREEN 1    ' cambia questo valore a seconda dell'adattatore
            ' usato

CIRCLE (20, 100), 20

FOR i% = 21 TO 299      ' si presuppone una risoluzione 320*200
      CIRCLE (i% - 1, 100), 20, 0 ' cancella il cerchio
                                  ' precedente
      CIRCLE (i%, 100), 20        ' disegna un nuovo cerchio

      FOR j% = 1 TO DELAY%        ' ritarda il loop
      NEXT j%

NEXT i%
```

CAPITOLO 12

1. a. Eseguire il programma.

   b. Osservare errori e punti critici in fase di esecuzione.
   c. Studiare le affermazioni che producono l'errore usando i listati, gli strumenti di programmazione e la vostra conoscenza della sintassi di QBasic.
   d. Correggere l'errore e provare il programma.

2. Un errore di sintassi è un errore di programmazione che viola le regole di QBasic. Un errore *run-time* è un errore che forza il programma a interrompersi inaspettatamente durante l'esecuzione.

3. Non vi è una risposta corretta; noi preferiamo i comandi Step e Toggle Breakpoint.

4. Il tasto funzione F7 esegue la riga corrente e quelle comprese tra la riga corrente e il cursore a piena velocità.

5. Imposta istruzione successiva imposta la riga in cui si trova il cursore come riga da eseguire successivamente.

6. Anche qui non c'è una risposta corretta; noi pensiamo che i peggiori siano gli errori logici provenienti da espressioni aritmetiche o condizionali sbagliate.

7. DIM e NEXT non sono scritti correttamente; la versione corretta è la seguente (ORSO.BAS):

```
' ORSO2.BAS
' Questo programma è la versione corretta di ORSO1.BAS

CLS

DIM orsi$(5)      ' dimensiona la matrice stringa
PRINT "Scrivete i nomi dei vostri cinque orsi preferiti."
PRINT

FOR i% = 1 TO 5  ' riceve le cinque stringhe
     INPUT "Orso: ", orsi$(i%)
NEXT i%
```



```
PRINT
PRINT "Avete inserito questi orsi:"
PRINT

FOR i% = 1 TO 5  ' stampa le cinque stringhe
     PRINT orsi%(i%)
NEXT i%
```

8. L'enunciato INPUT dovrebbe essere un enunciato LINE INPUT per gestire la virgola nell'input dell'utente e l'espressione condizionale nell'enunciato IF dovrebbe usare un operatore maggiore di (>) invece di uno minore di (<). Questa è la versione corretta (NOME2.BAS):

```
' NOME2.BAS
' Questo programma separa nomi e cognomi e li stampa.

CLS

PRINT "Scrivete nome e cognome in questo formato: ";
PRINT "Cognome, Nome"
PRINT

LINE INPUT "Nome: ", nomeIntero$

virgolaPosiz% = INSTR(1, nomeIntero$, ",")

IF (virgolaPosiz% > 0) THEN
   Cognome$ = LEFT$(nomeIntero$, virgolaPosiz% - 1)
   Nome$ = RIGHT$(nomeIntero$, LEN(nomeIntero$) - virgolaPosiz% - 1)

   PRINT
   PRINT "Che bel nome! Piacere, "
   PRINT Nome$; " "; Cognome$; "!"
ELSE
   PRINT
   PRINT "Nome non nel formato Cognome, Nome."
END IF
```